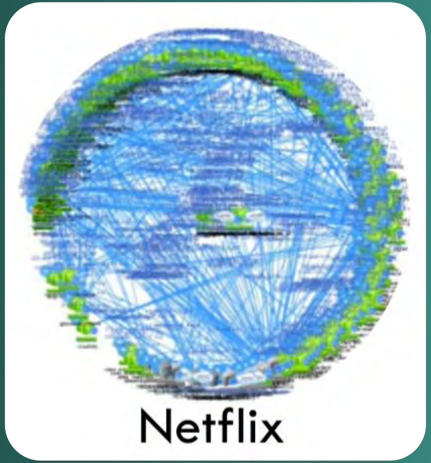
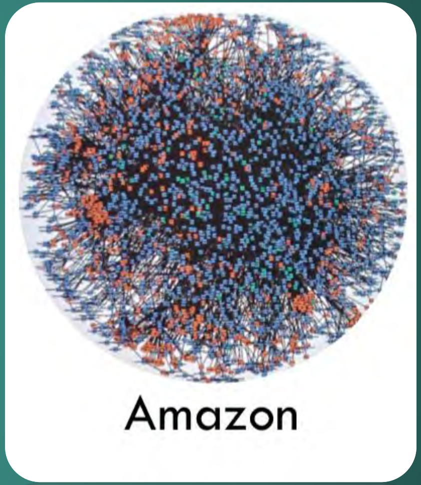
Aspects of Microservice Interactions

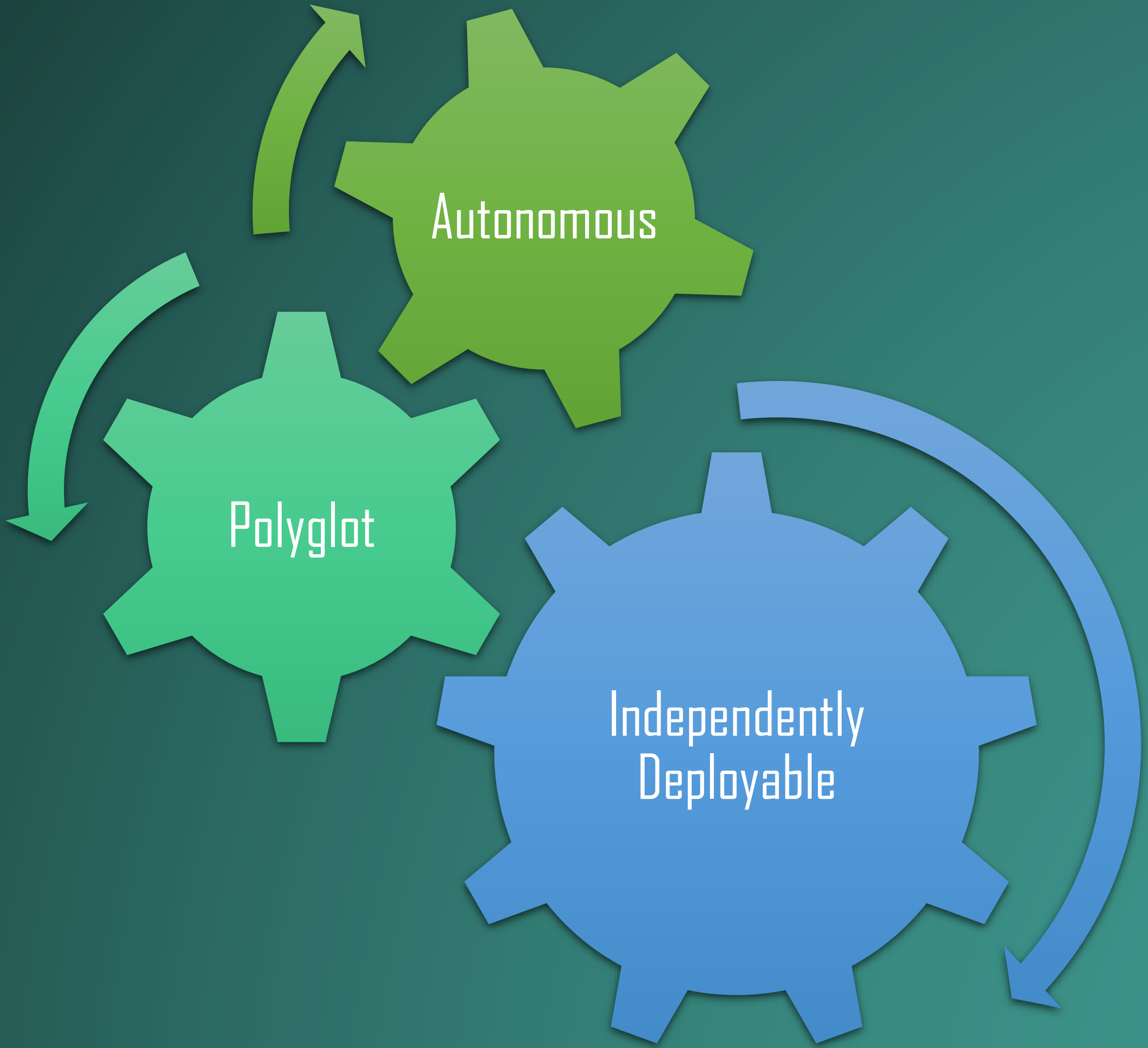
Death Star Architecture





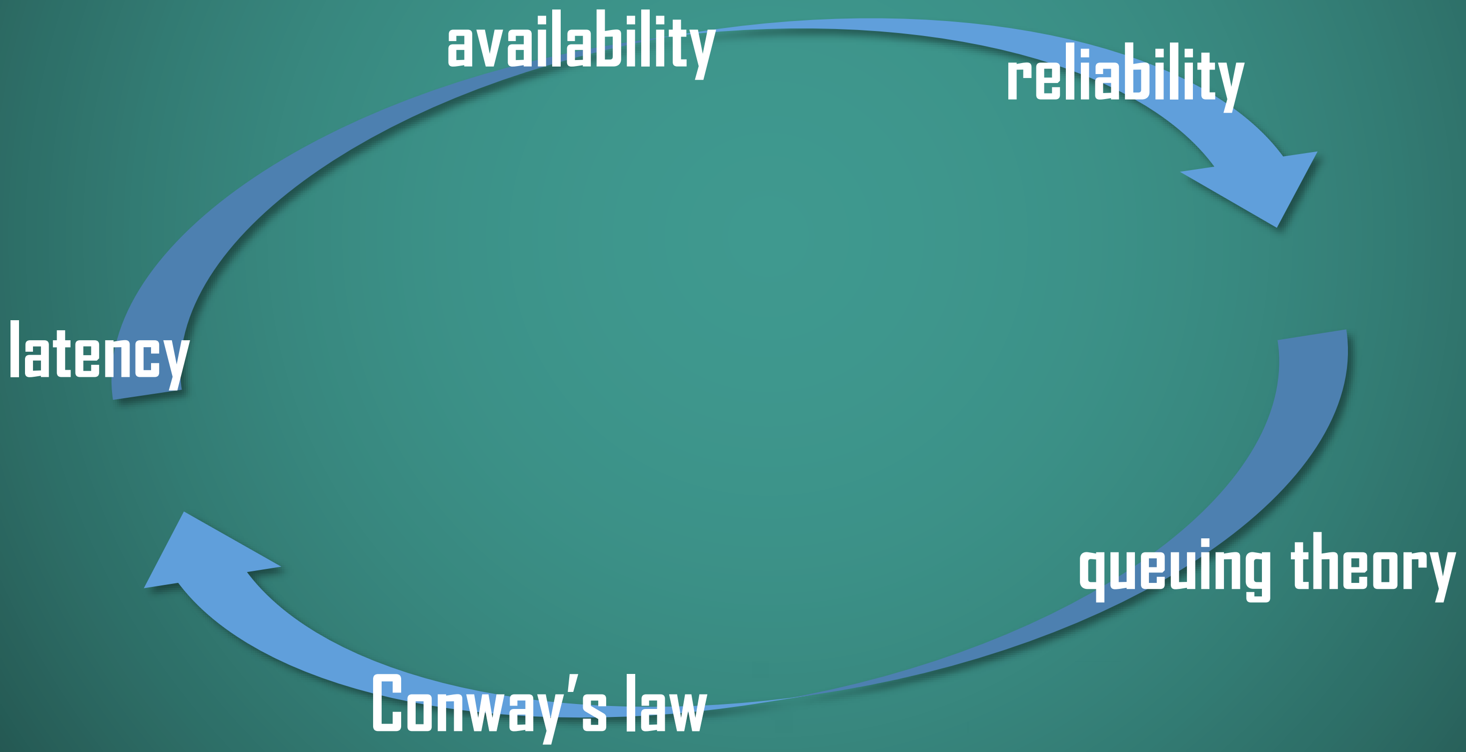
WHY?

Characteristics

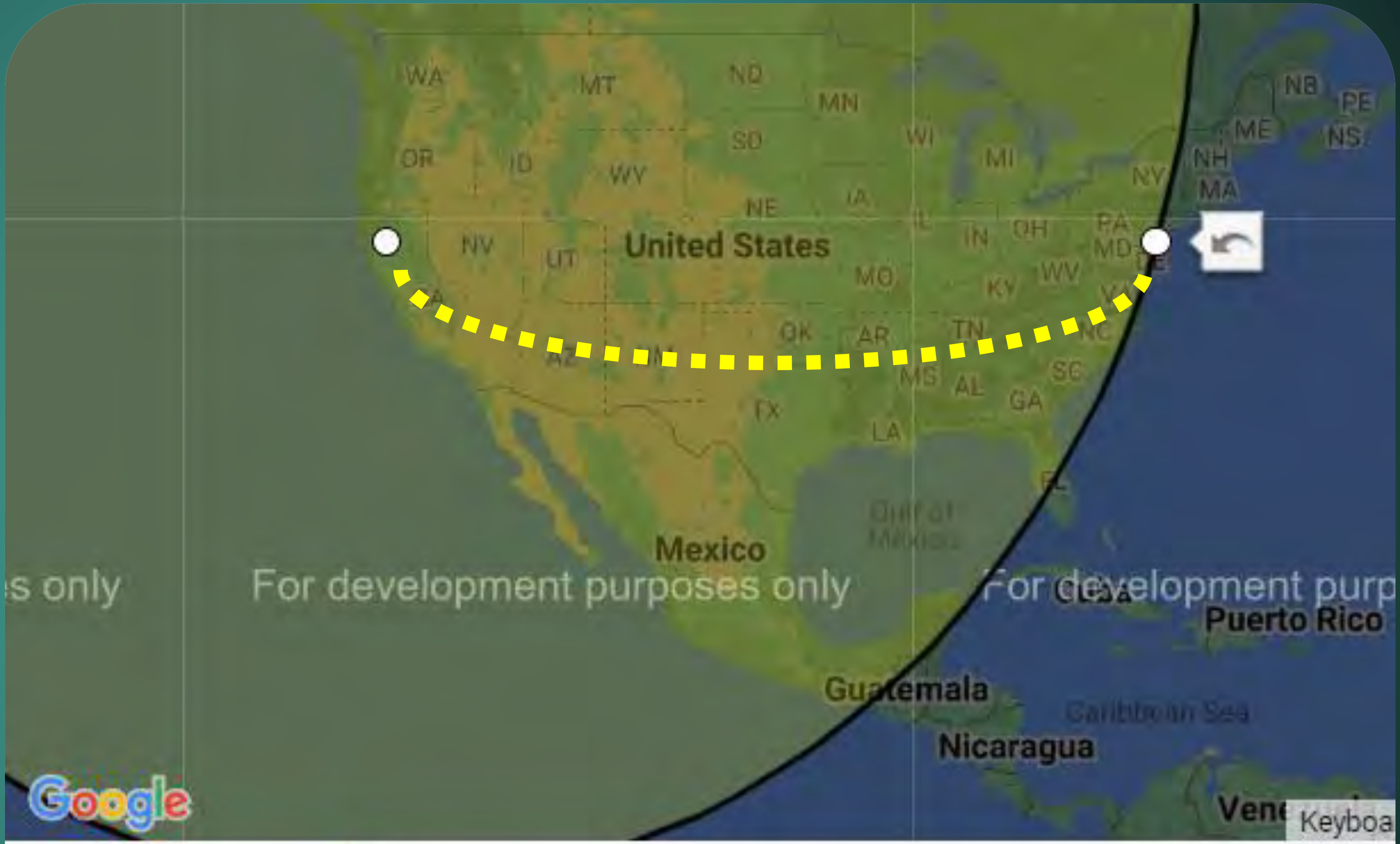


NETWORK
CALL

Driving Forces

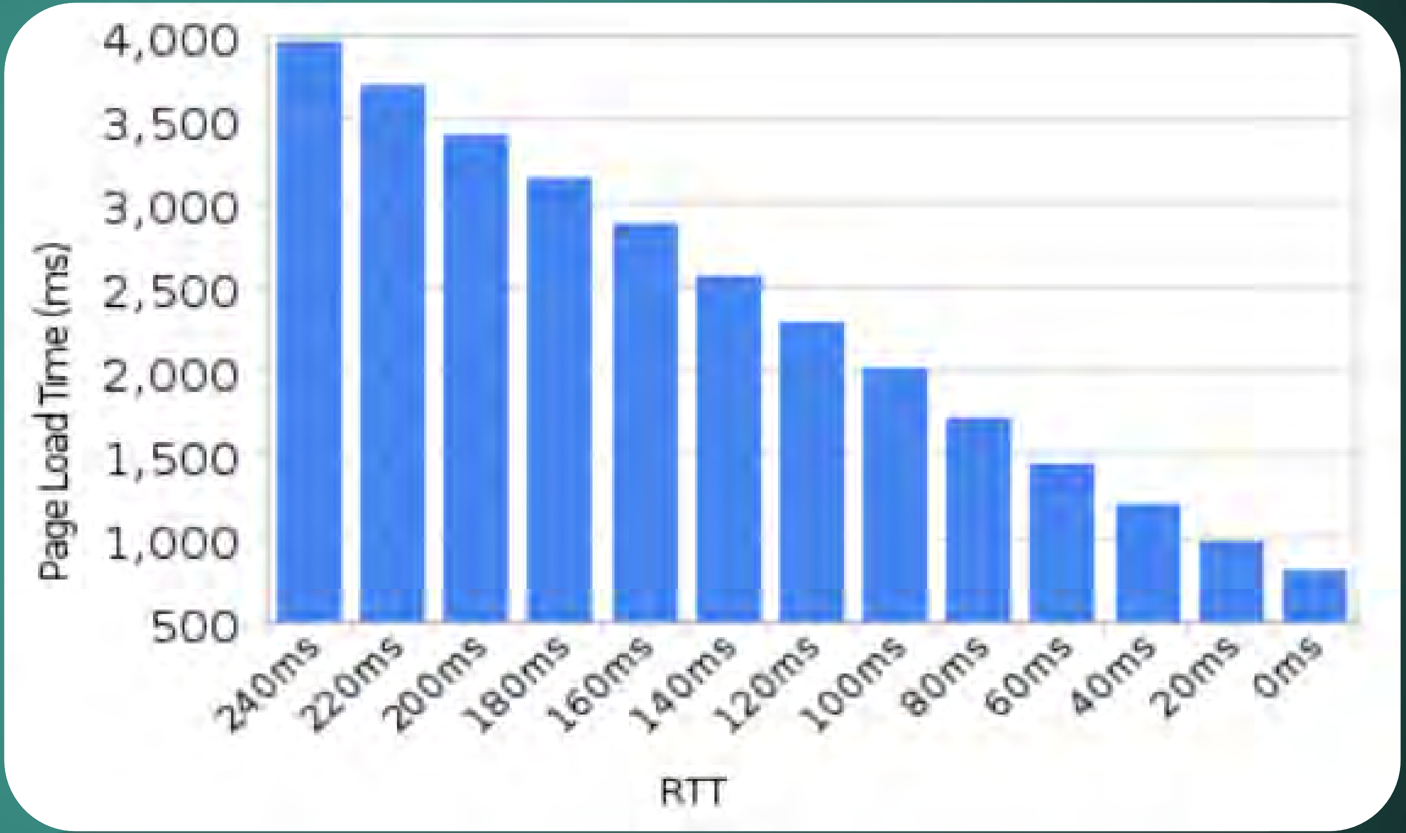


Latency



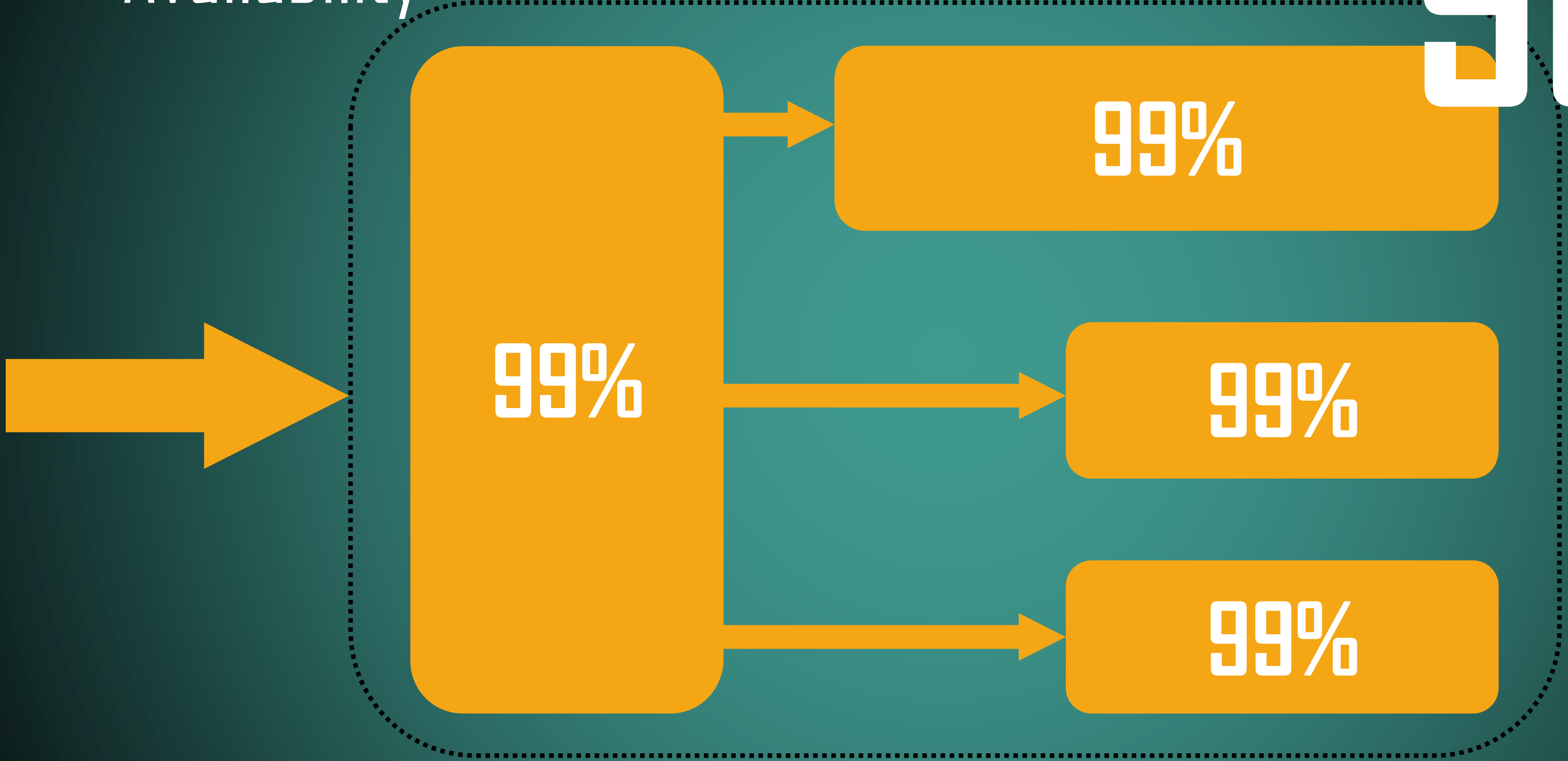
Radius: **4008**km / **2490.46**mi, RTT in vacuum: **27**ms, RTT in fiber: **41.04**ms.

Latency


Page Load Time (ms)
4,000
3,500
3,000
2,500
2,000
1,500
1,000
500
240ms
220ms
200ms
180ms
160ms
140ms
120ms
100ms
80ms
60ms
40ms
20ms
0ms
RTT

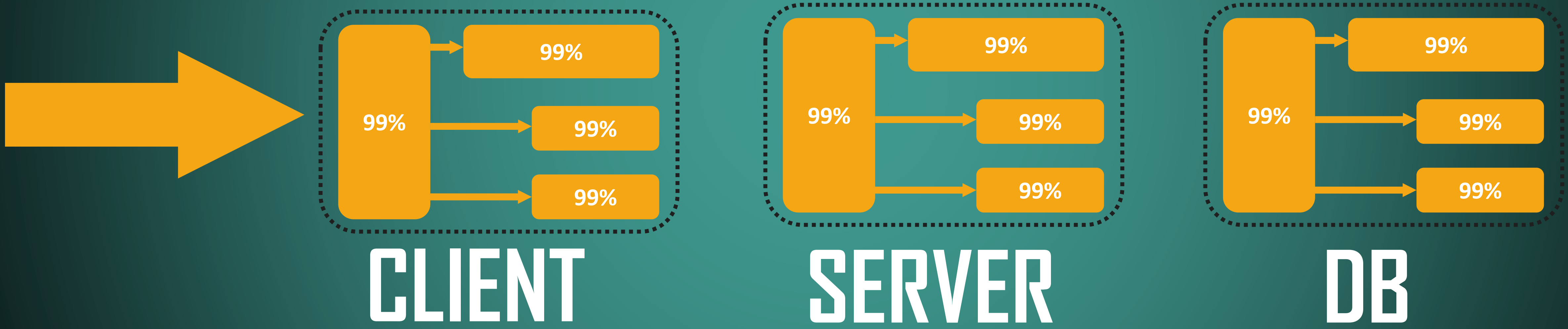
Availability

96%

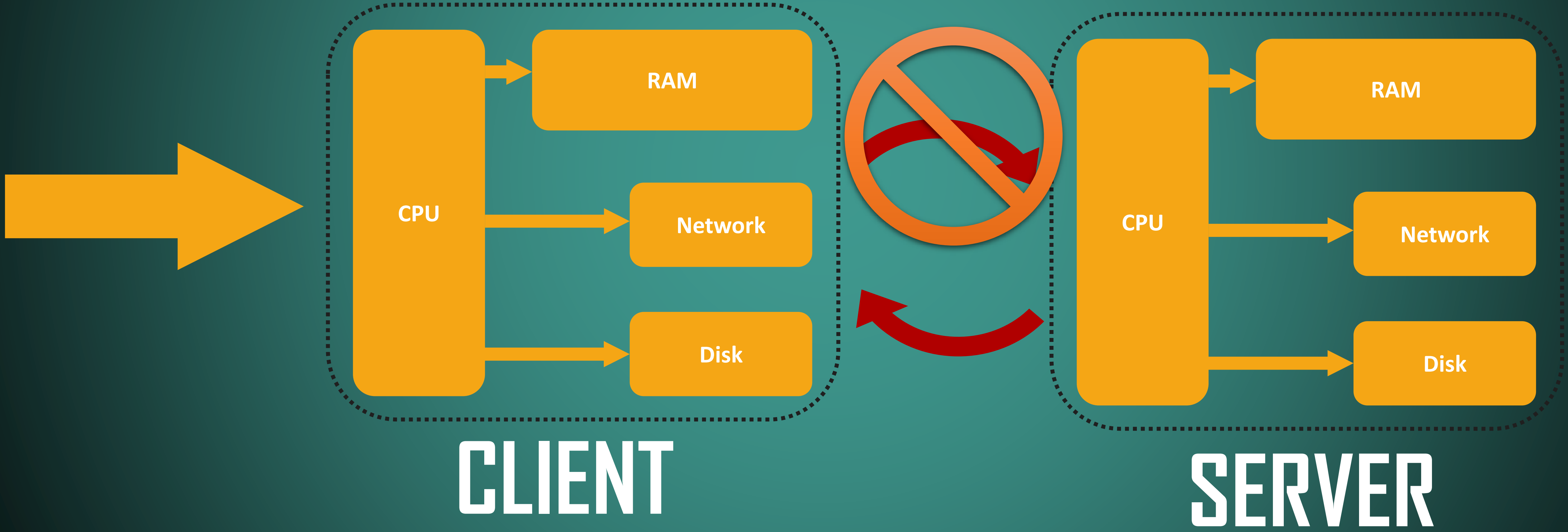


Availability

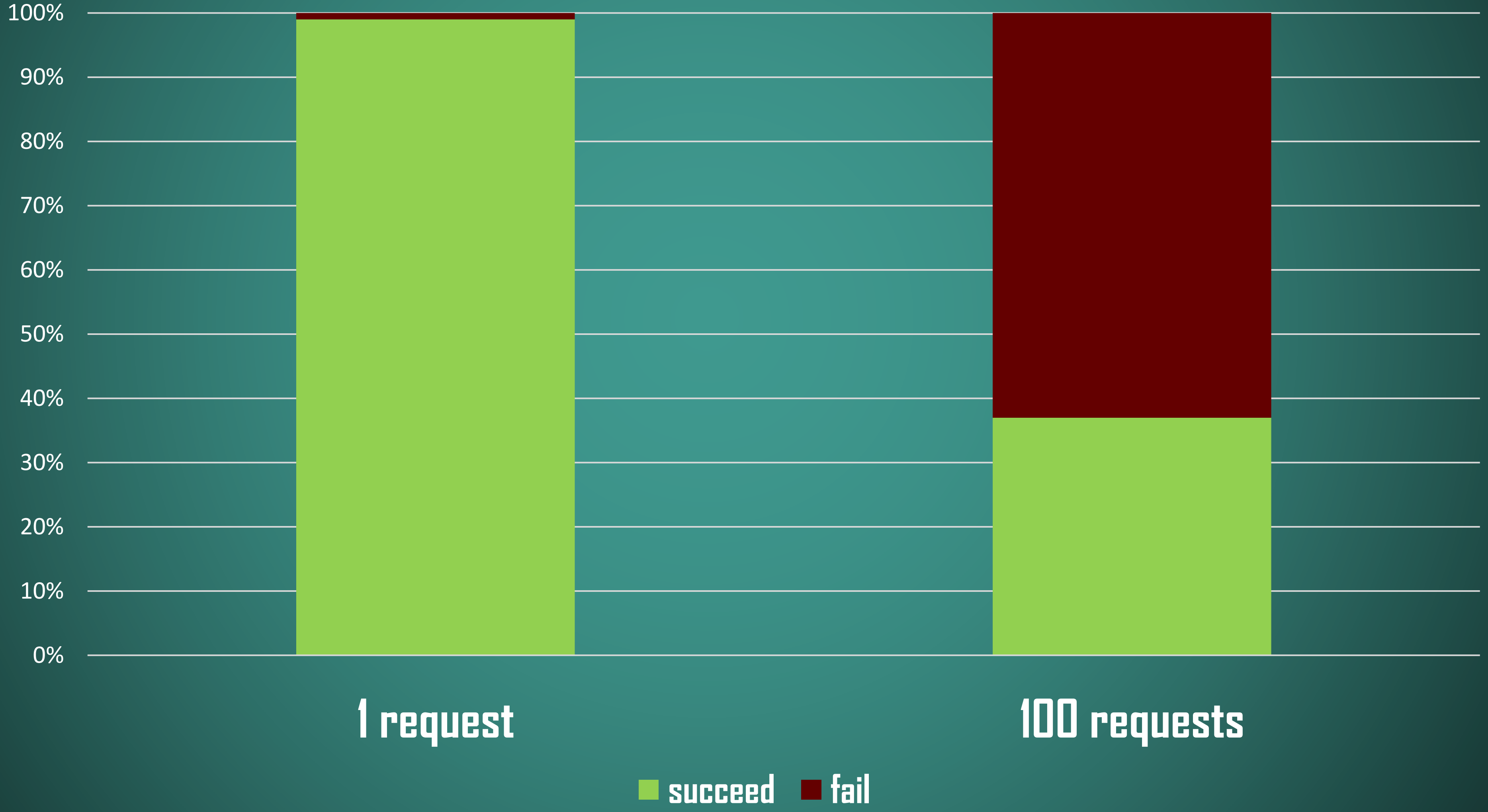
88.5%



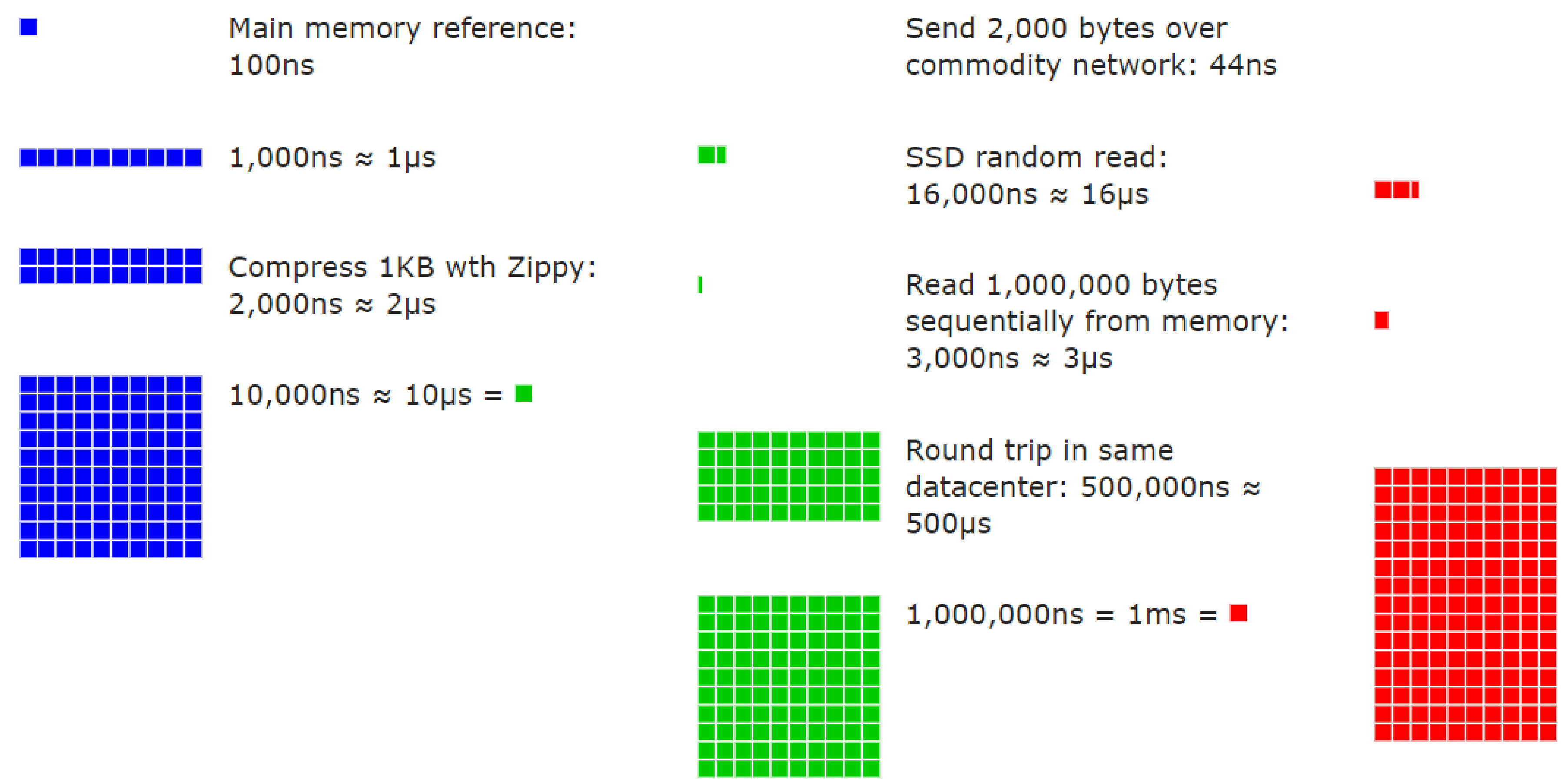
Reliability



Reliability



Latency Numbers Every Programmer Should Know



Main memory reference: 100ns

1,000ns ≈ 1µs

Compress 1KB wth Zippy: 2,000ns ≈ 2µs

10,000ns ≈ 10µs = 🟩

Send 2,000 bytes over commodity network: 44ns

SSD random read: 16,000ns ≈ 16µs

Read 1,000,000 bytes sequentially from memory: 3,000ns ≈ 3µs

Round trip in same datacenter: 500,000ns ≈ 500µs

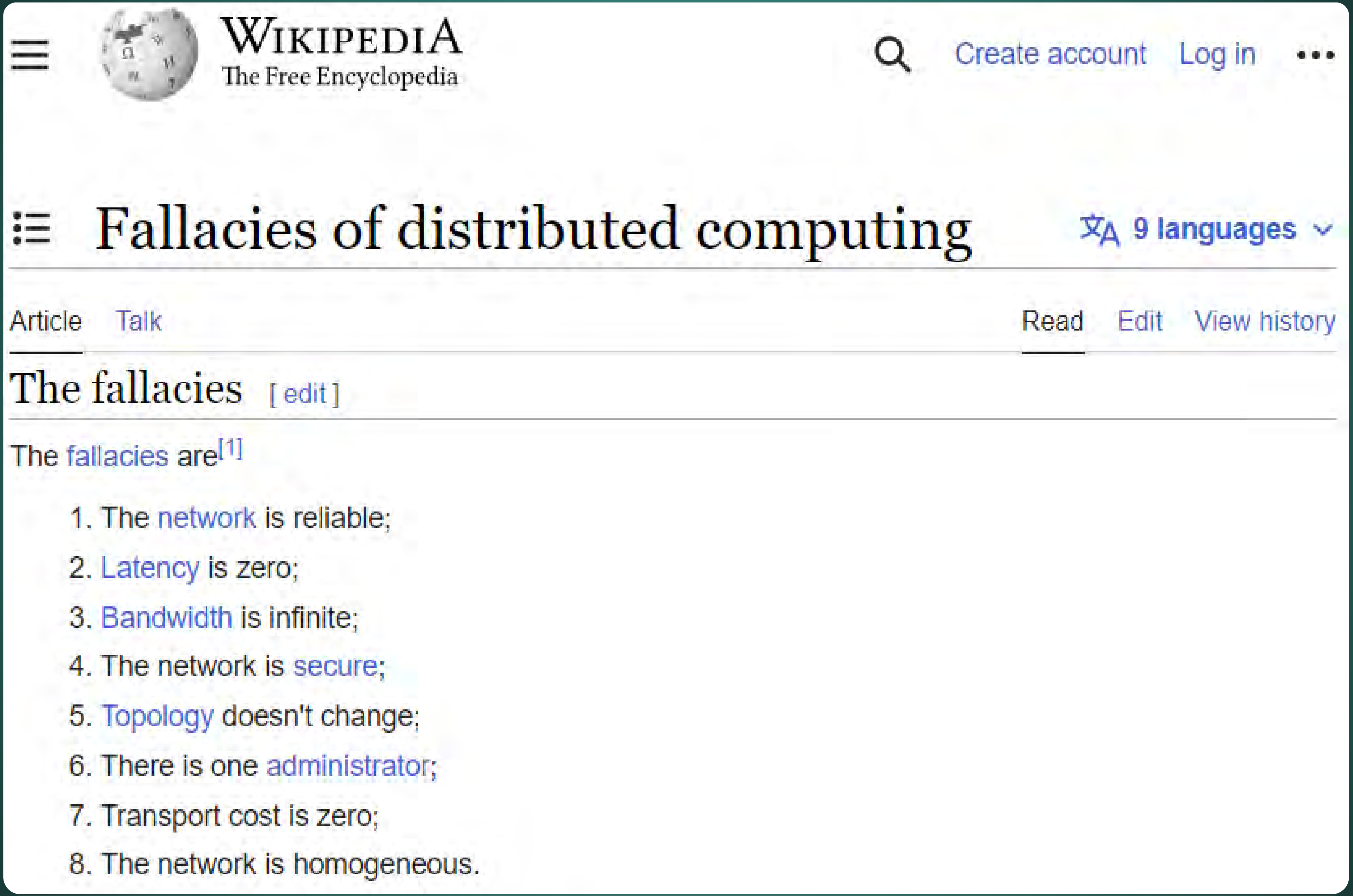
1,000,000ns = 1ms = 🟥

Read 1,000,000 bytes sequentially from SSD: 49,000ns ≈ 49µs

Disk seek: 2,000,000ns ≈ 2ms

Read 1,000,000 bytes sequentially from disk: 825,000ns ≈ 825µs

Packet roundtrip CA to Netherlands: 150,000,000ns ≈ 150ms



Wikipedia
The Free Encyclopedia

Create account Log in

Fallacies of distributed computing

9 languages

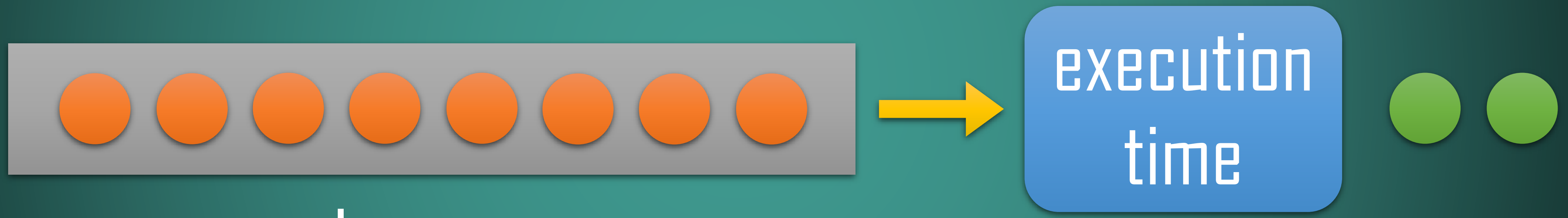
Article Talk Read Edit View history

The fallacies [edit]

The fallacies are[1]

1. The network is reliable;
2. Latency is zero;
3. Bandwidth is infinite;
4. The network is secure;
5. Topology doesn't change;
6. There is one administrator;
7. Transport cost is zero;
8. The network is homogeneous.

Queuing Theory



latency

arrival rate

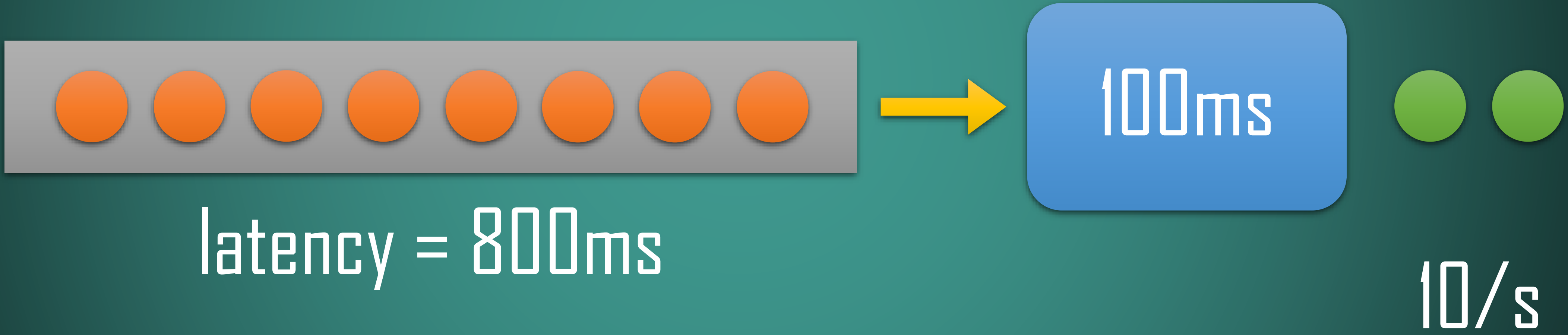
departure rate

Queuing Theory

arrival rate > departure rate

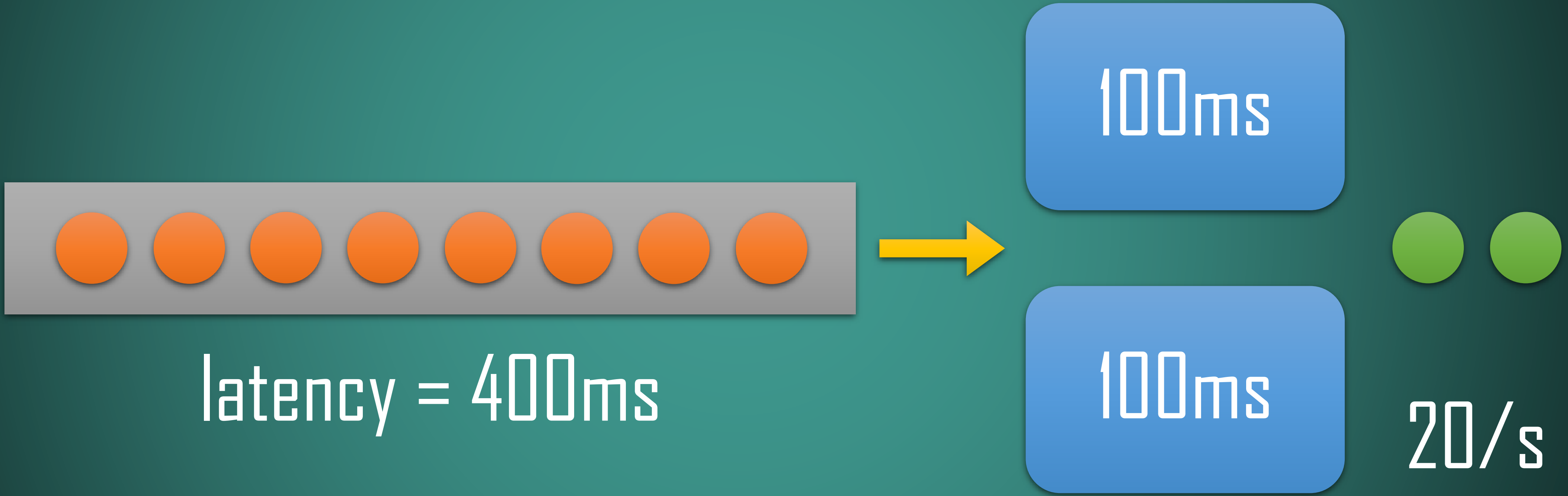
backpressure!

Queuing Theory



latency = 800ms

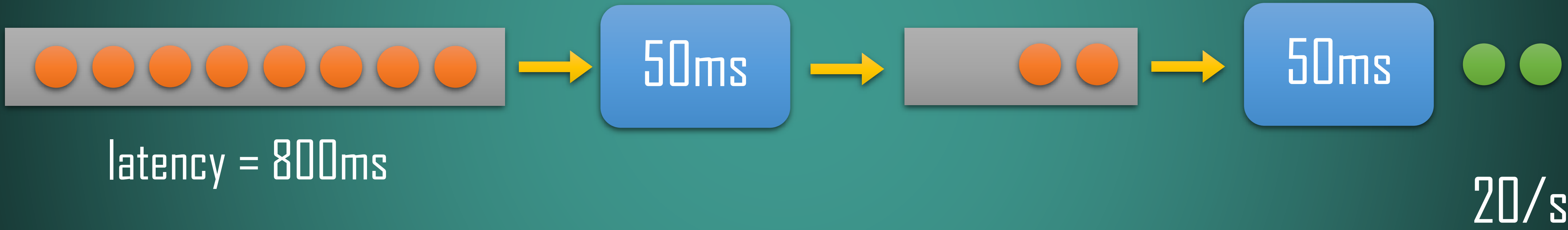
Queuing Theory



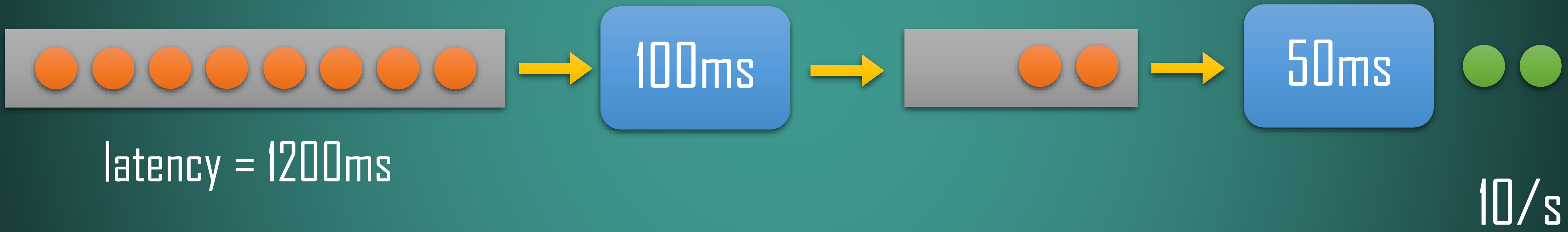
latency = 400ms

20/s

Queuing Theory



Queuing Theory

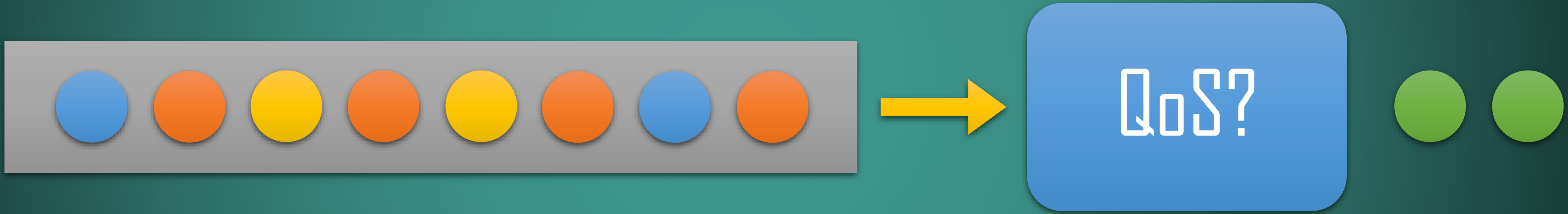


Queuing Theory

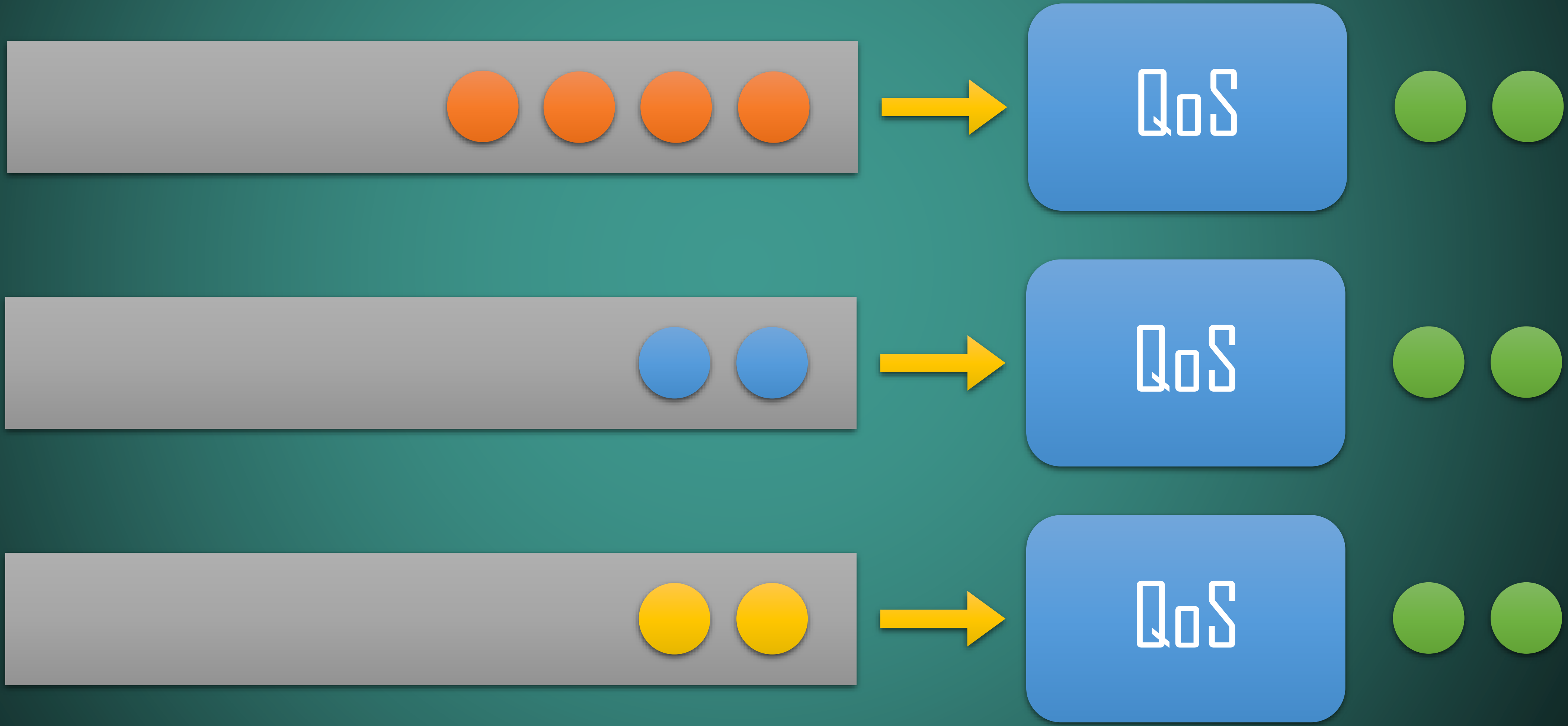
```
latency = sum(execution time / parallelism) * queue length
```

```
throughput = min((parallelism / execution time))
```

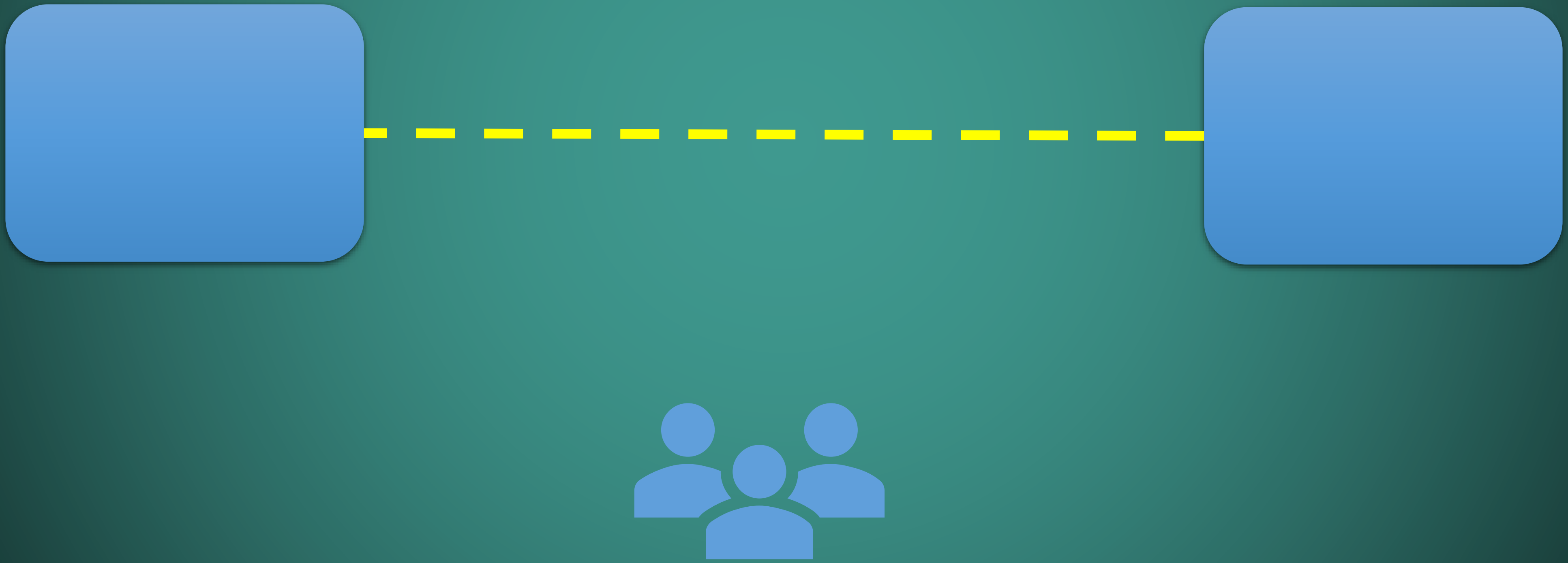
Queuing Theory



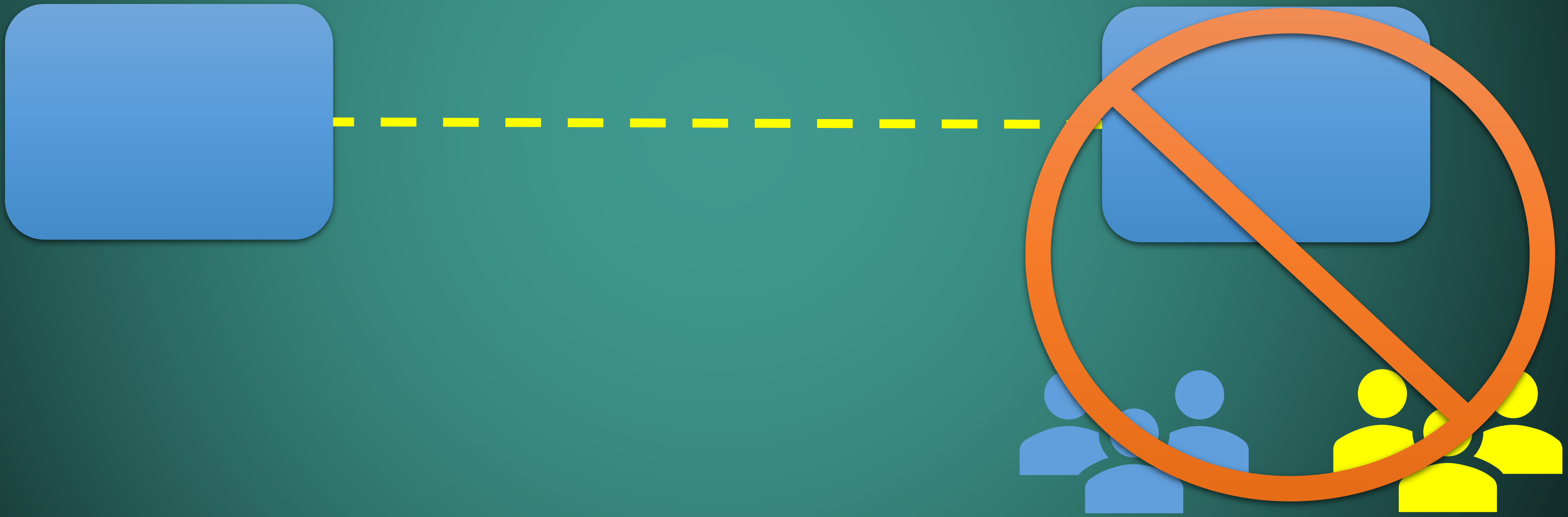
Queuing Theory



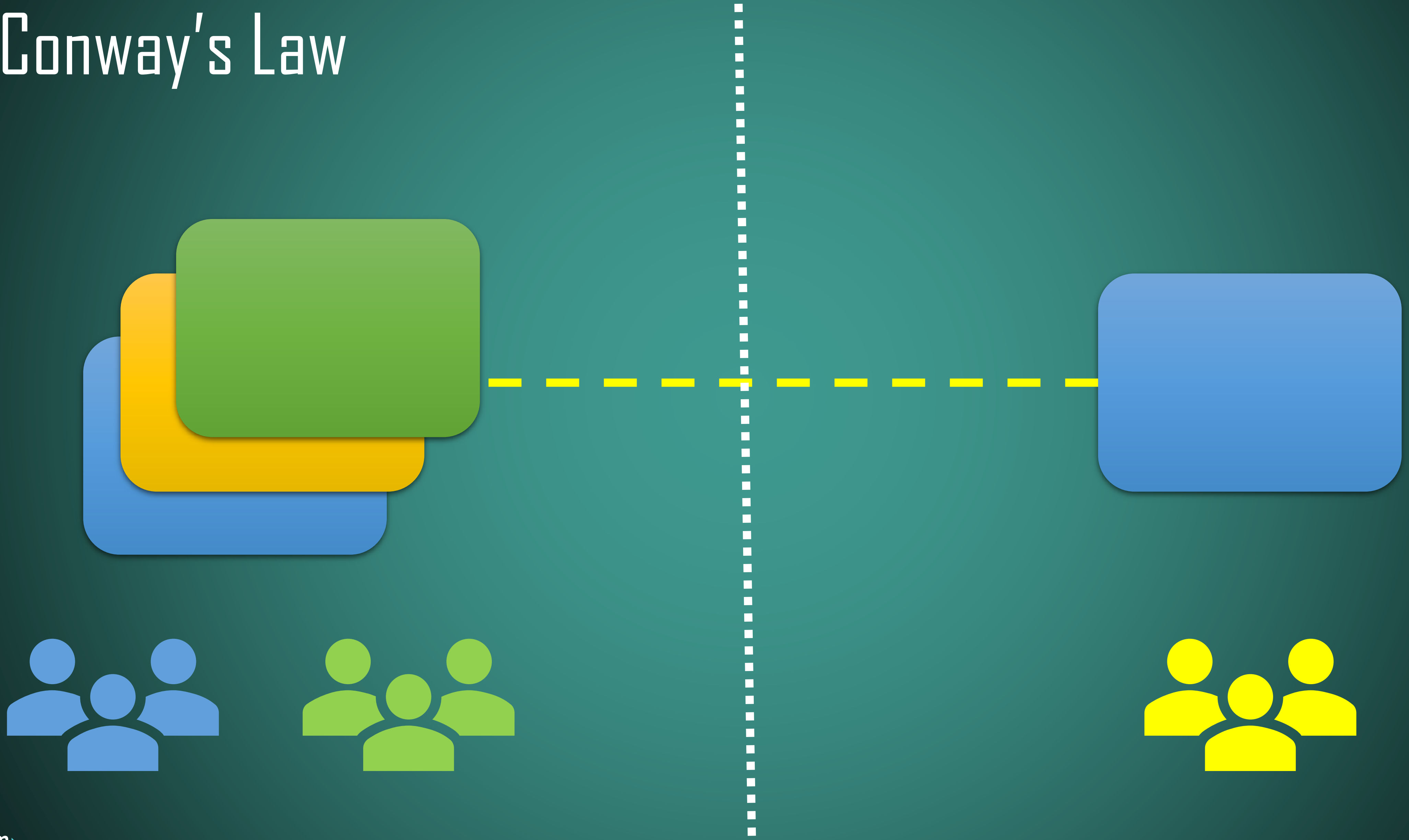
Conway's Law



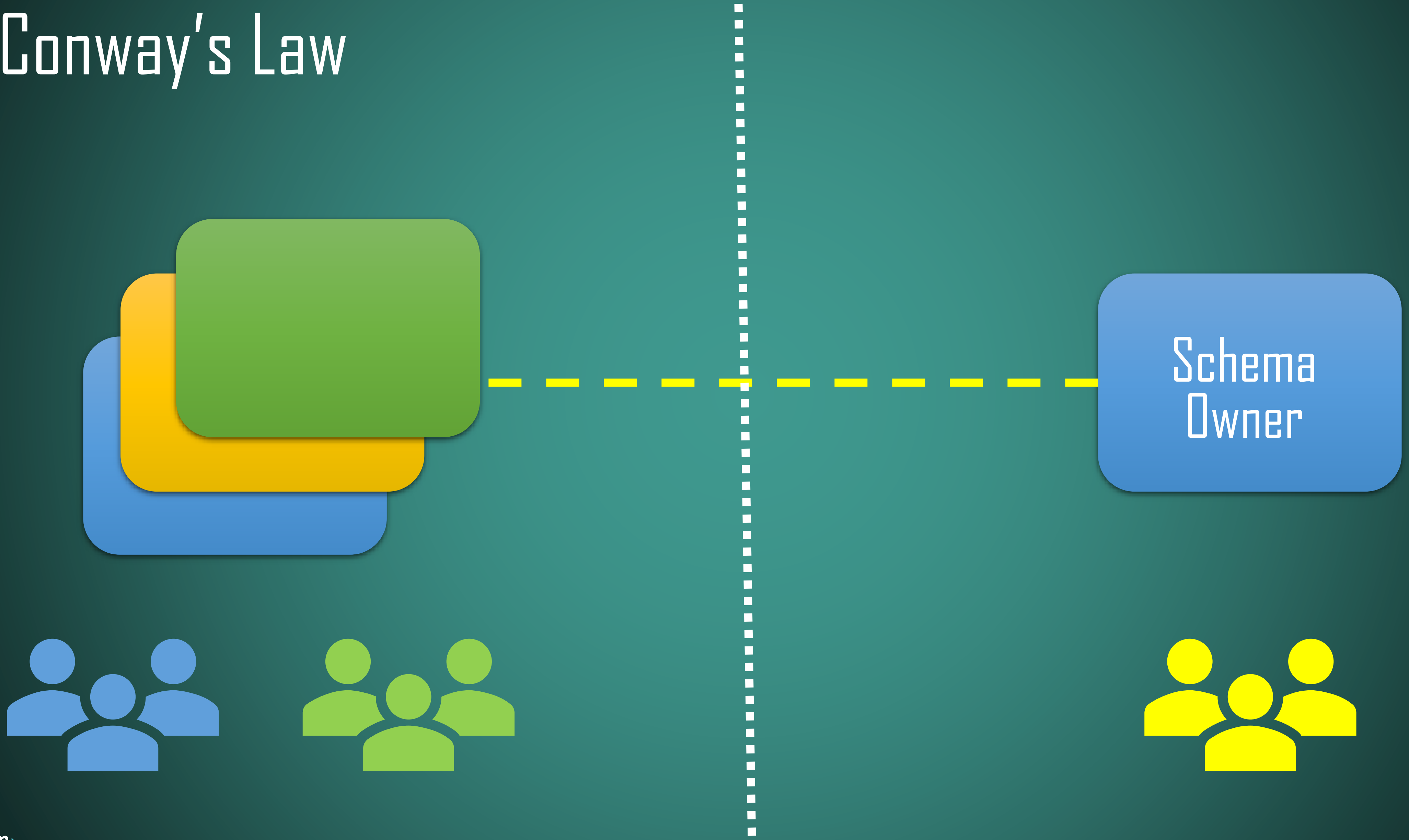
Conway's Law



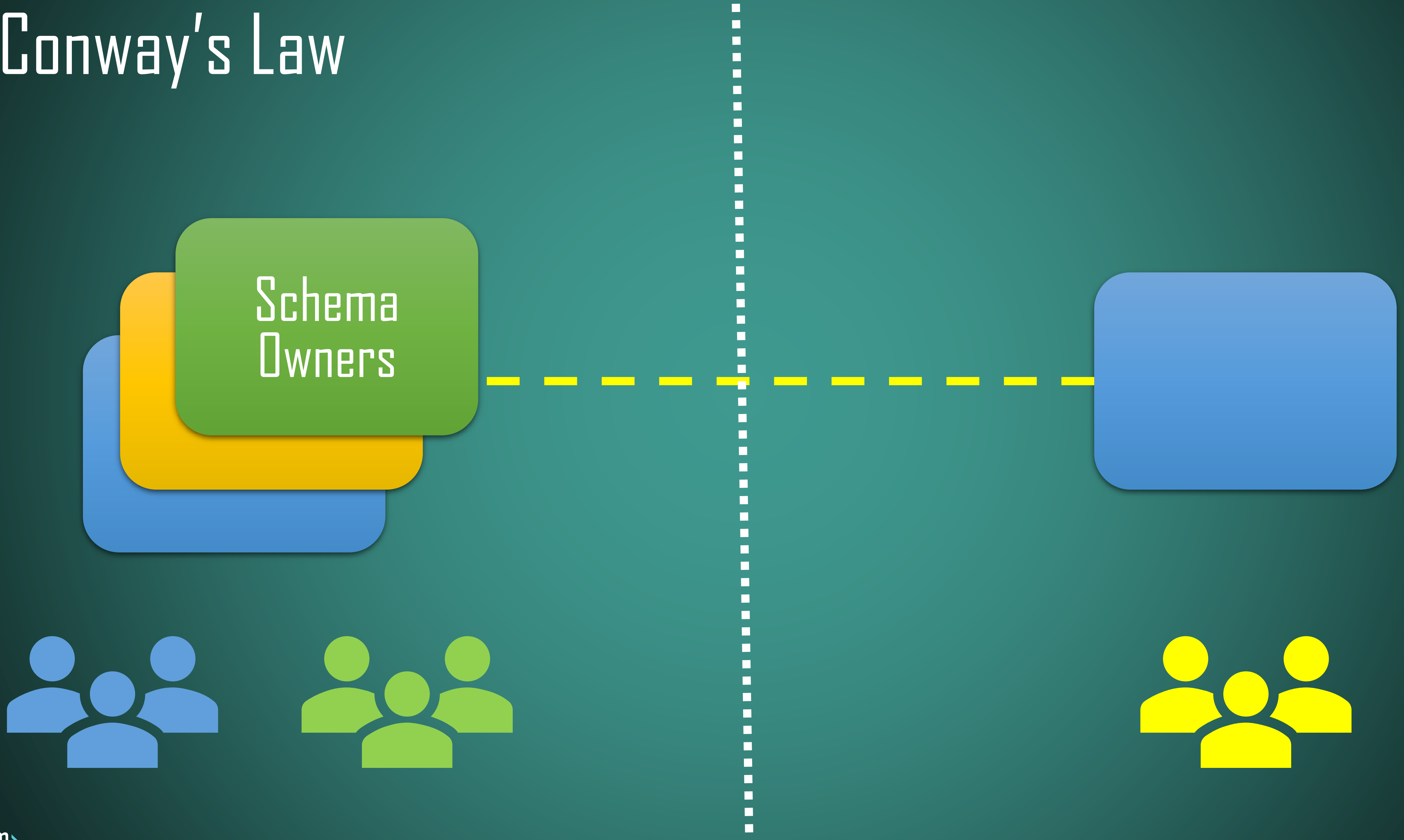
Conway's Law



Conway's Law

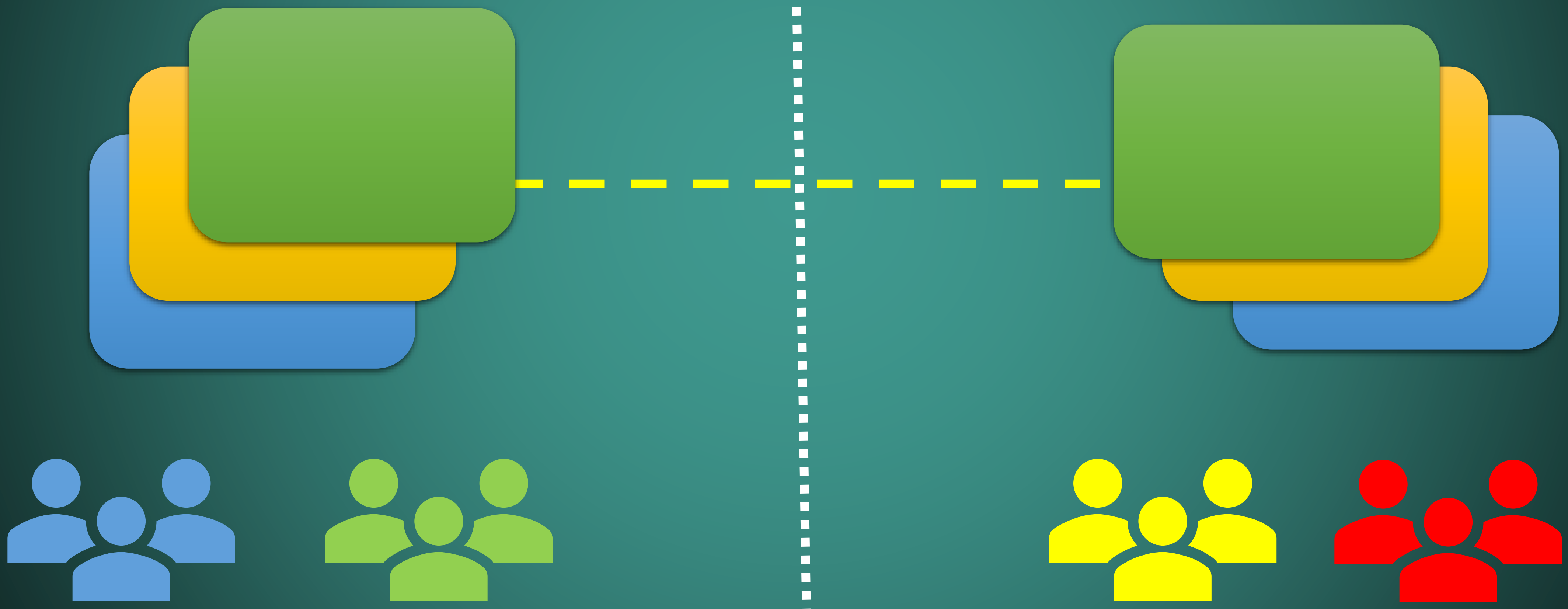


Conway's Law



Conway's Law

Schema Federation



<epam>

TOOLBOX

CQRS

SCHEMAS

CONTRACT-BASED TESTS

CACHING

MULTI-WRITES

SYNC TO ASYNC

POLLING, FAKE BOUNDARIES

CLOUD NATIVE

AUTO-SCALING

BACKPRESSURE

SAGA

ORCHESTRATION, COREOGRAPHY

12 FACTOR

SERVICE MESH

TECHNOLOGY

GRPC, REST, GRAPHQL

MESSAGING

CORRELATION ID, ROUTING SLIP, EXACTLY ONCE
TOPICS VS QUEUES

CONCURRENCY MODEL

REACTIVE, ACTOR-BASED, COROUTINES

RESILIENCY PATTERNS

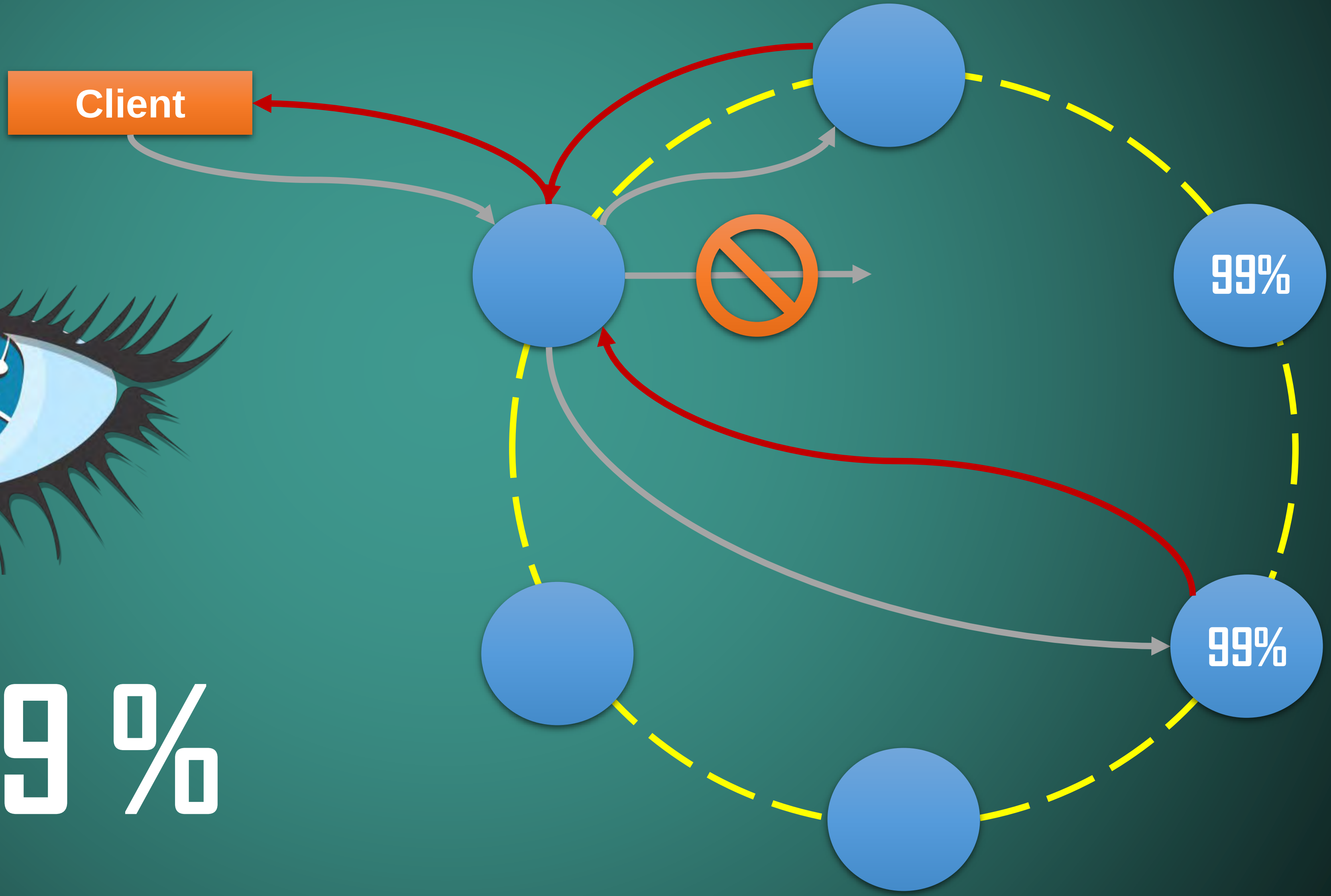
CIRCUIT BREAKER, BULKHEAD, RETRIES, TIMEOUTS

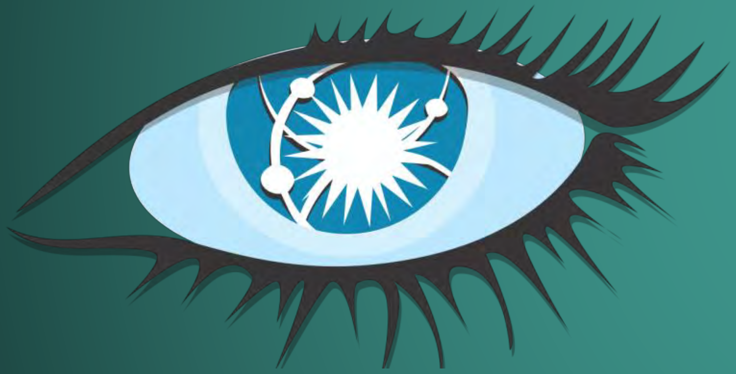
OBSERVABILITY

SLO, TRACING, METRICS, LOGS

EXAMPLES

Rapid Read Protection

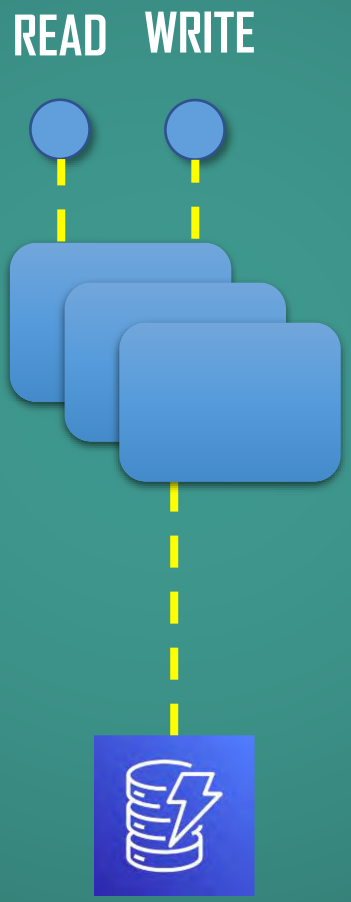




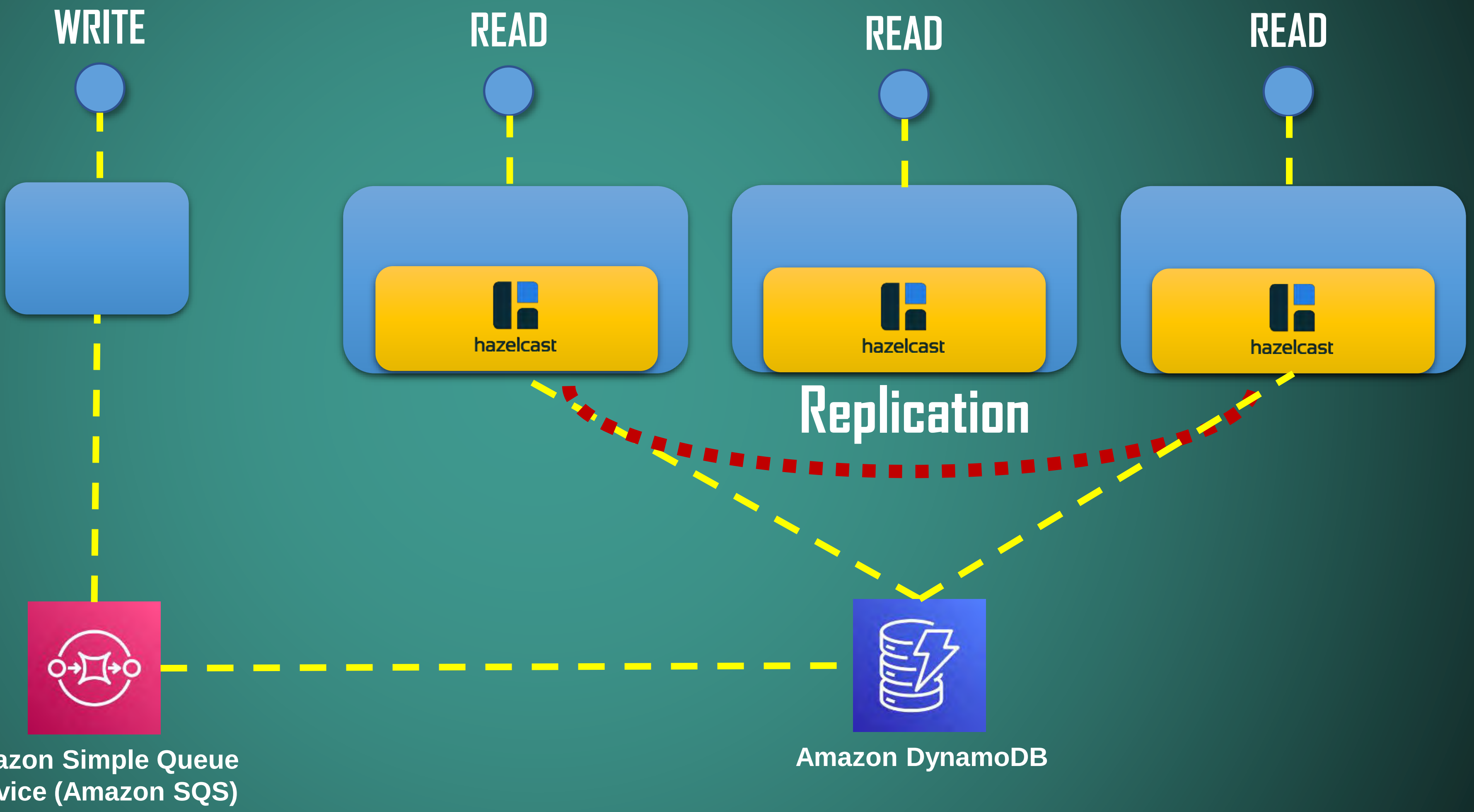
~ 99.99 %

- Read-heavy workload
- Immutable writes
- Low latency baseline
- Cold Cache & Distributed

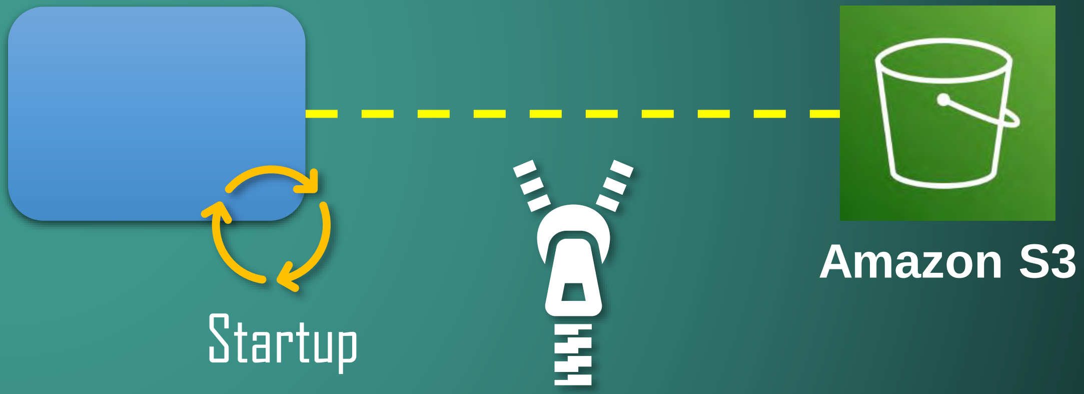
  Cache won't work



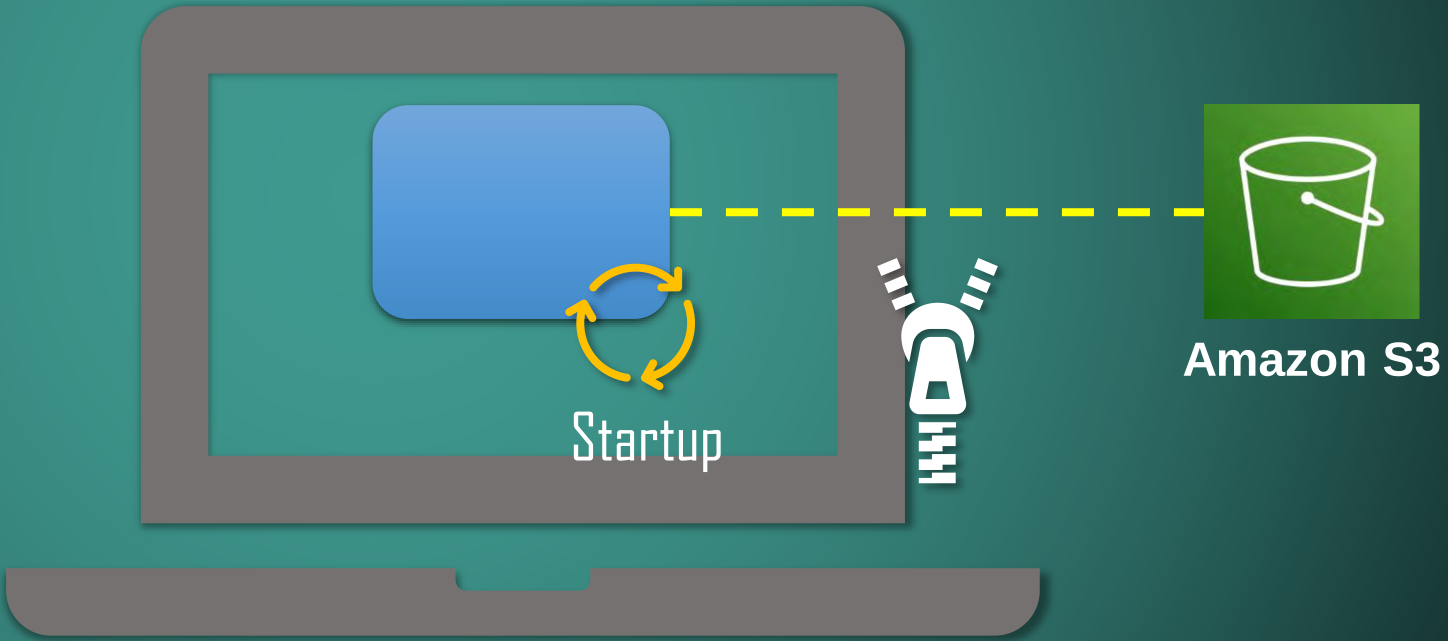
CQRS


WRITE
READ
READ
READ
hazelcast
hazelcast
hazelcast
Replication
Amazon Simple Queue Service (Amazon SQS)
Amazon DynamoDB

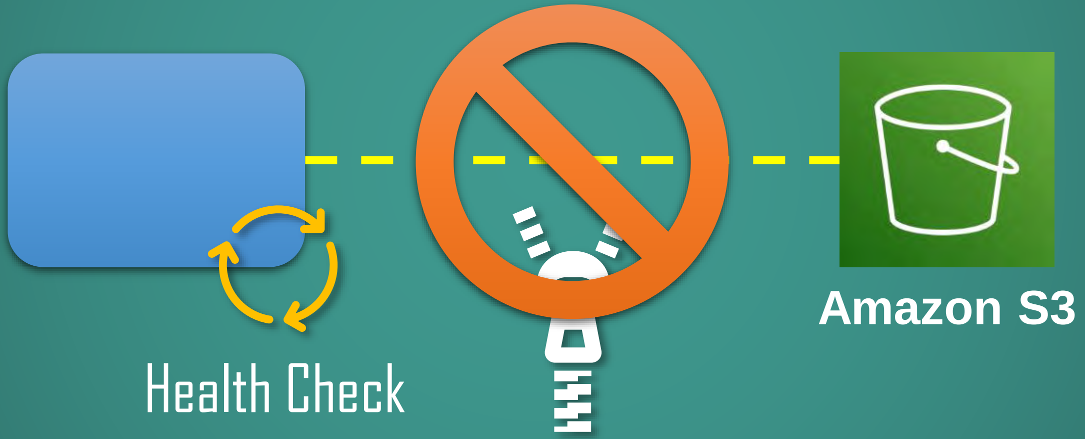
Client Libraries



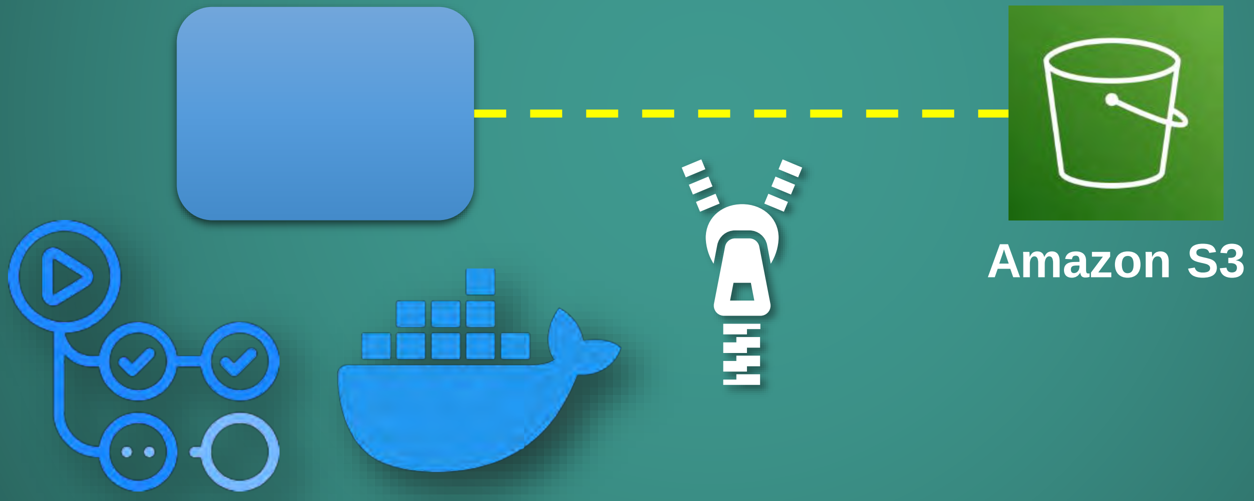
Client Libraries



Client Libraries

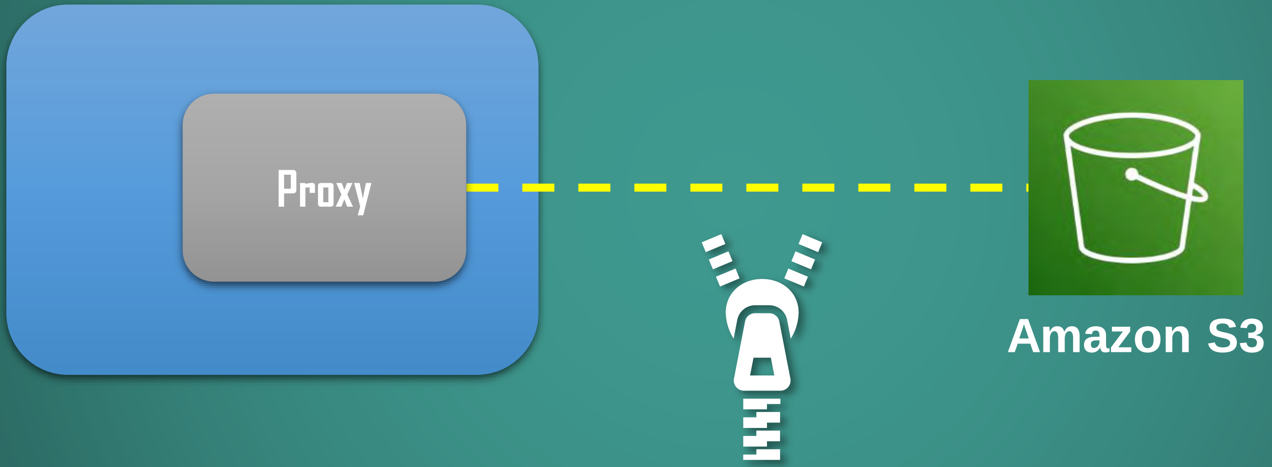


Shift Left

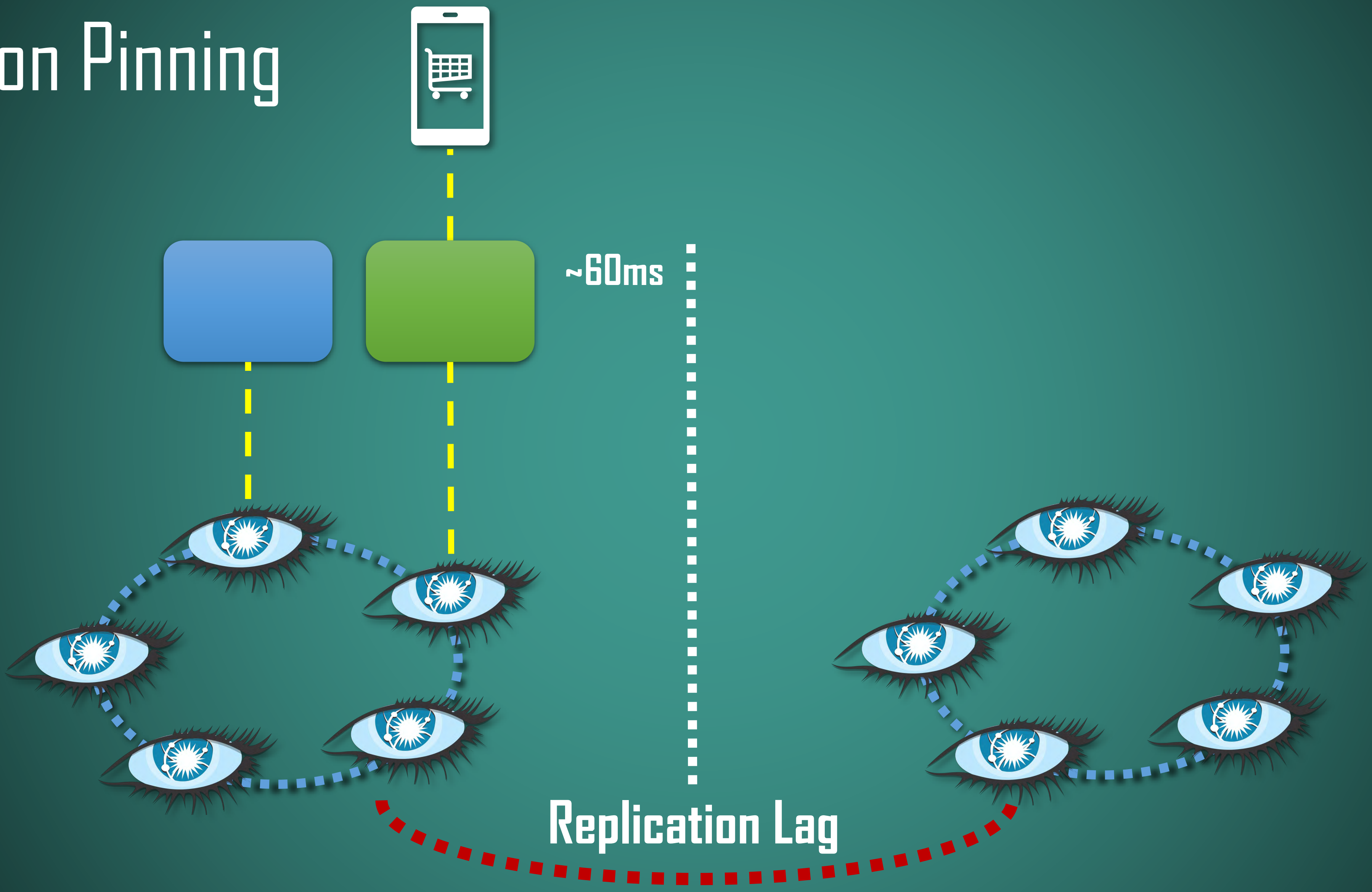


CI/CD

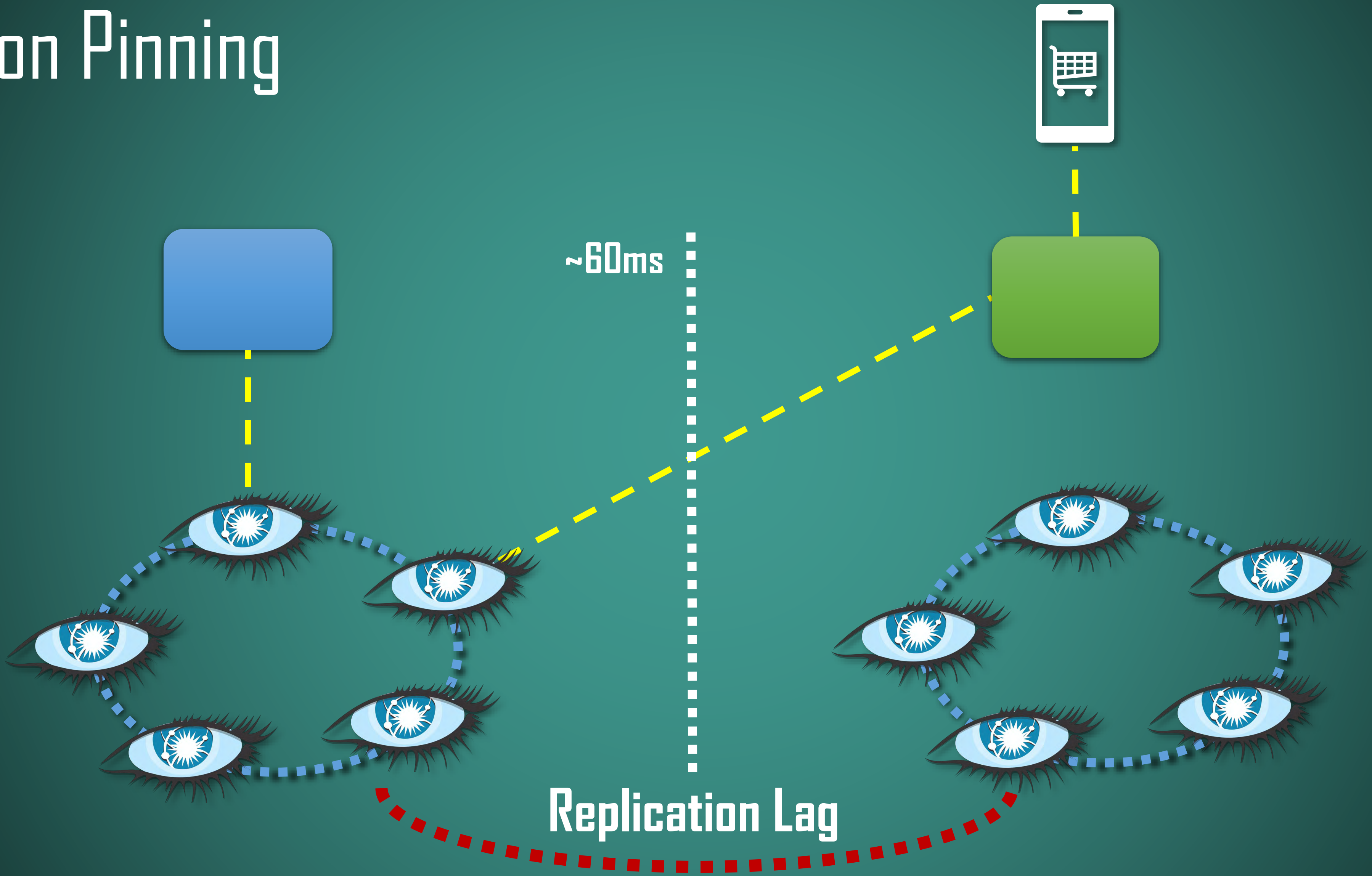
Proxy



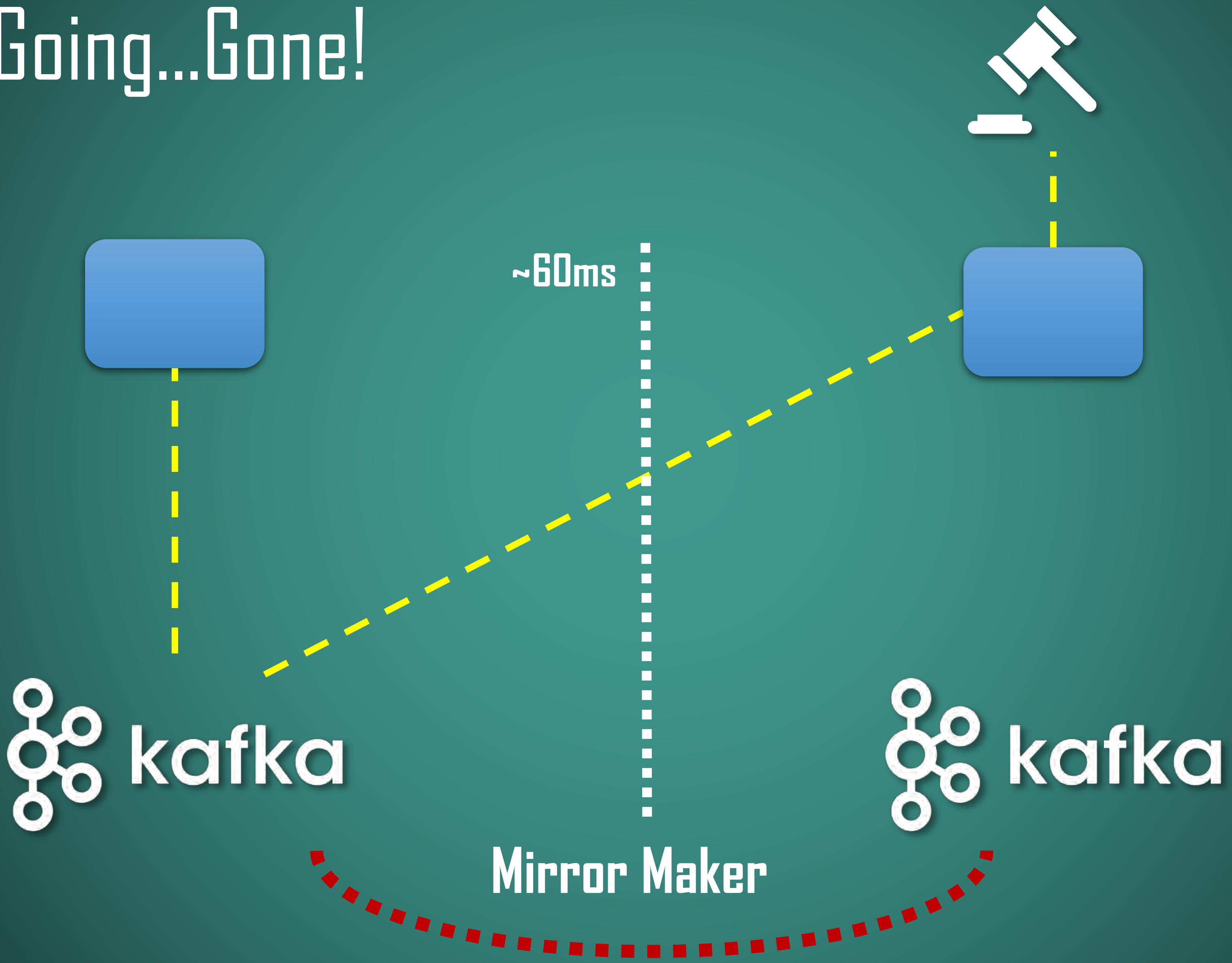
Region Pinning

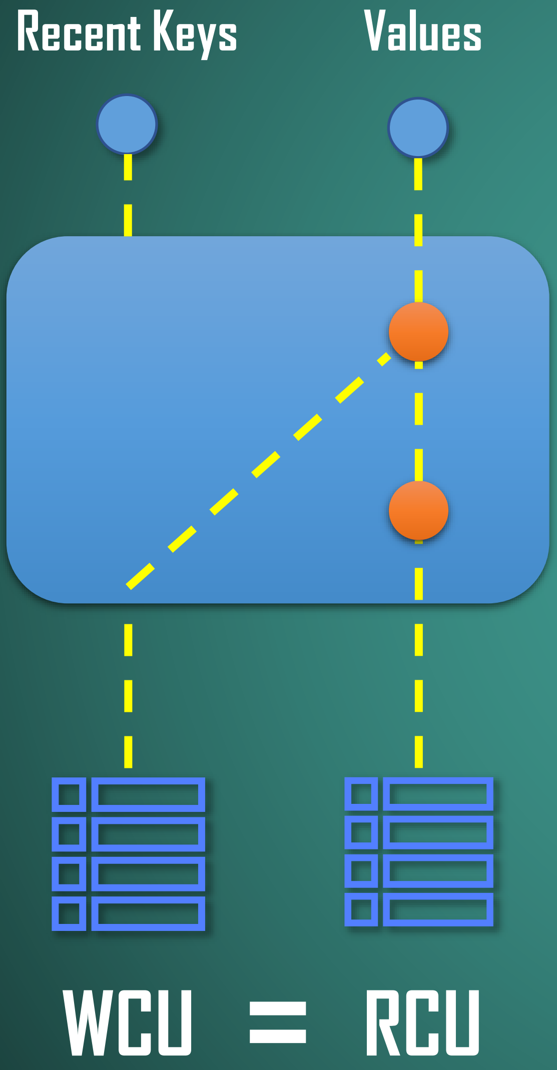


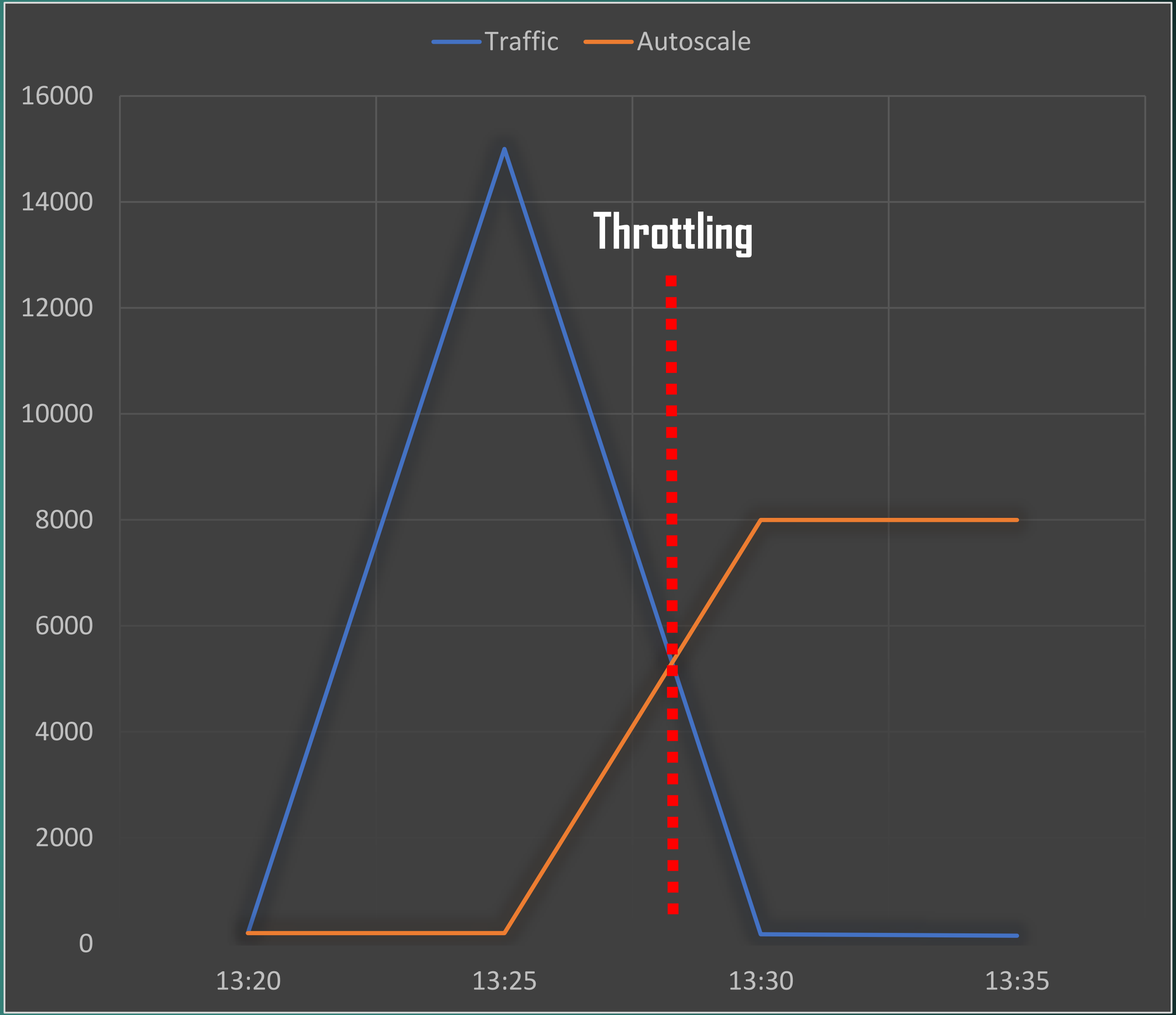
Region Pinning



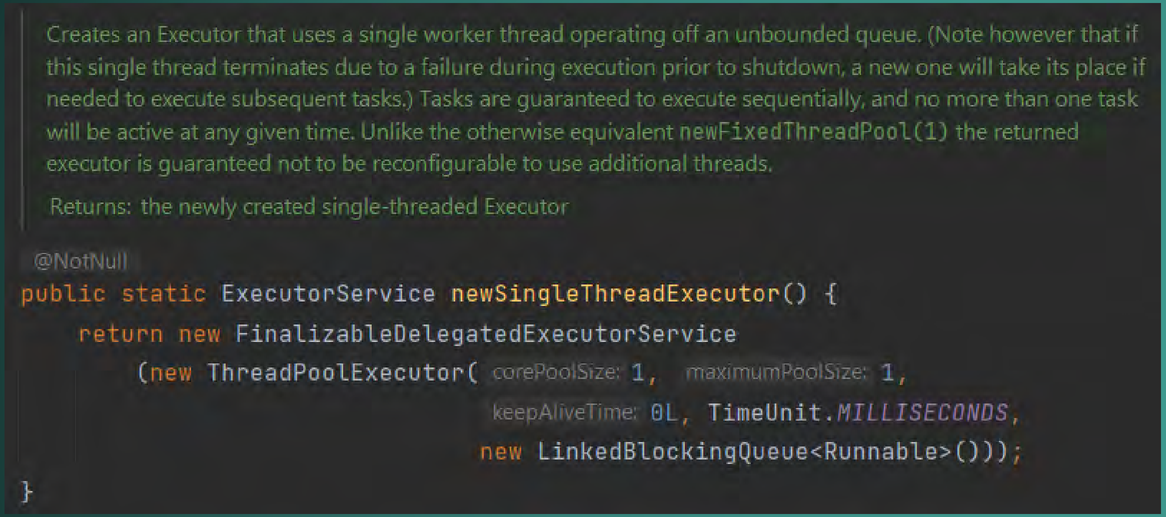
Going...Going...Gone!




Recent Keys
Values
WCU = RCU


Traffic
Autoscale
Throttling
16000
14000
12000
10000
8000
6000
4000
2000
0
13:20
13:25
13:30
13:35

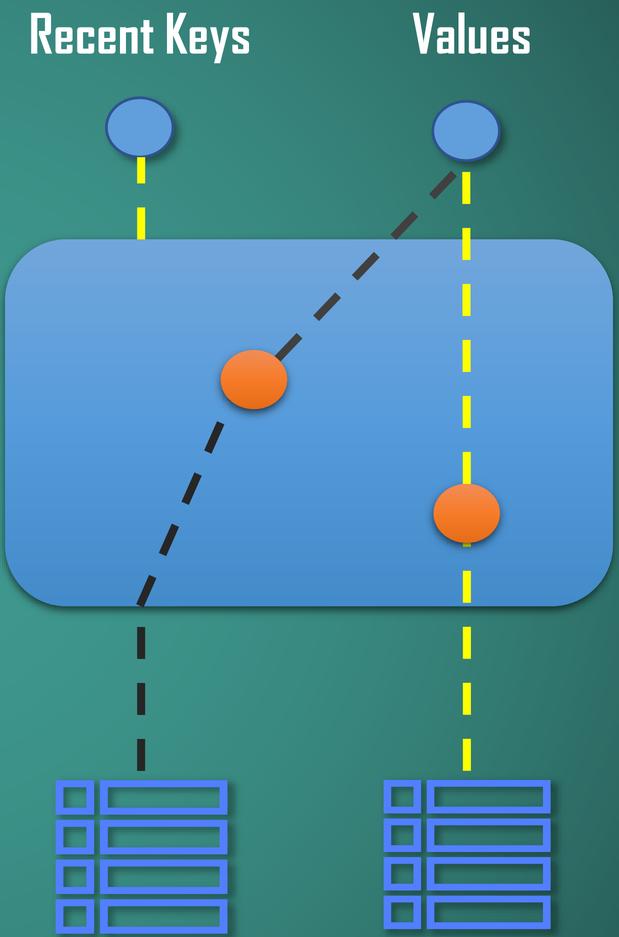
Separate Critical Path



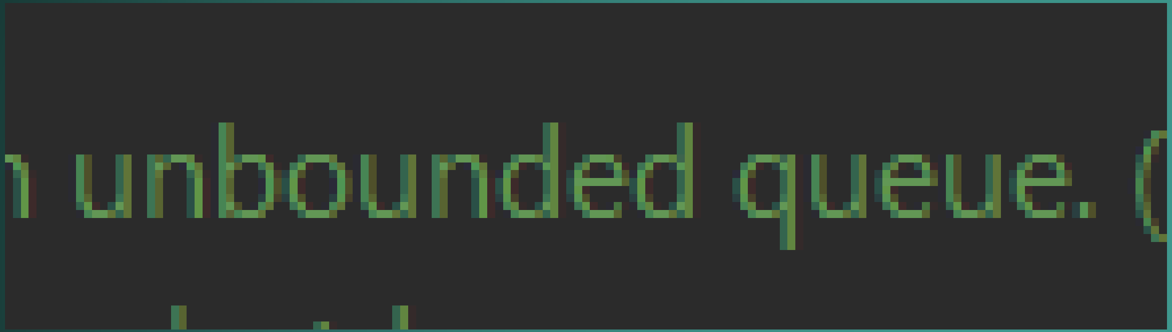
Creates an Executor that uses a single worker thread operating off an unbounded queue. (Note however that if this single thread terminates due to a failure during execution prior to shutdown, a new one will take its place if needed to execute subsequent tasks.) Tasks are guaranteed to execute sequentially, and no more than one task will be active at any given time. Unlike the otherwise equivalent `newFixedThreadPool(1)` the returned executor is guaranteed not to be reconfigurable to use additional threads.

Returns: the newly created single-threaded Executor

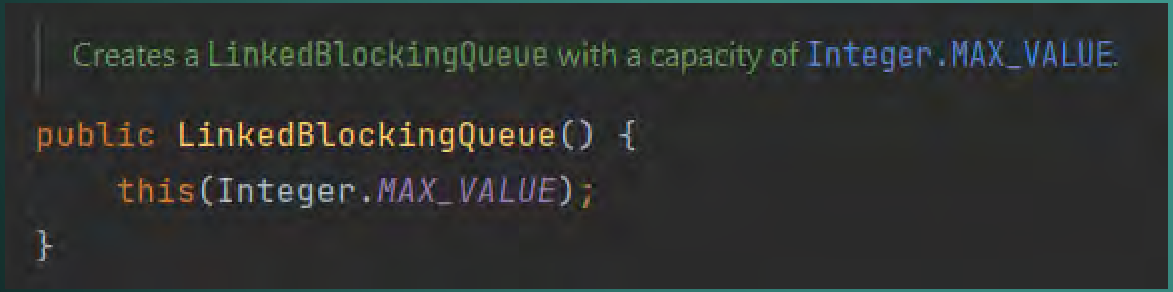
```
@NotNull
public static ExecutorService newSingleThreadExecutor() {
    return new FinalizableDelegatedExecutorService
        (new ThreadPoolExecutor( corePoolSize: 1,  maximumPoolSize: 1,
                                keepAliveTime: 0L, TimeUnit.MILLISECONDS,
                                new LinkedBlockingQueue<Runnable>()));
}
```



Separate Critical Path

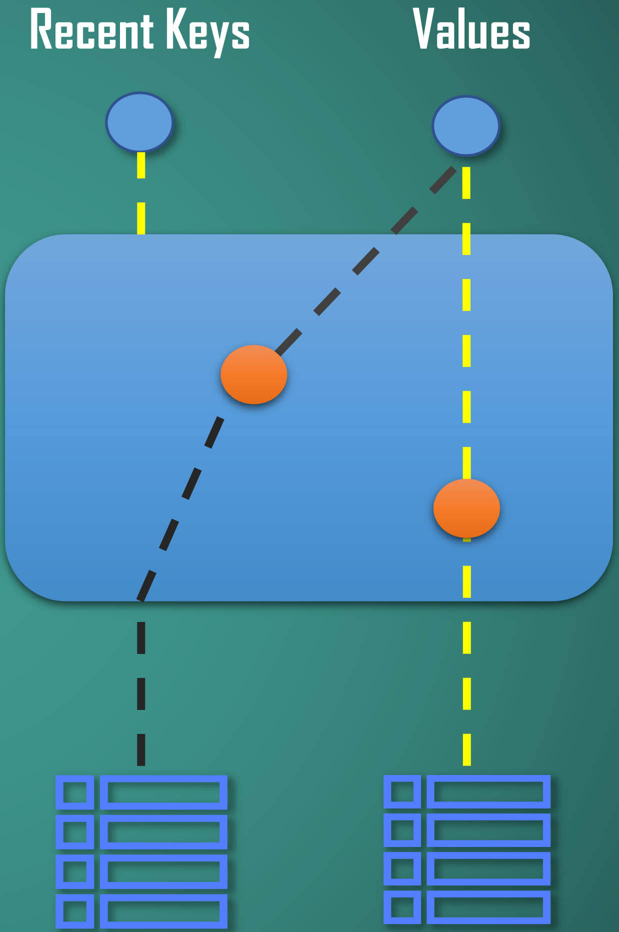


unbounded queue.



Creates a LinkedBlockingQueue with a capacity of Integer.MAX_VALUE.

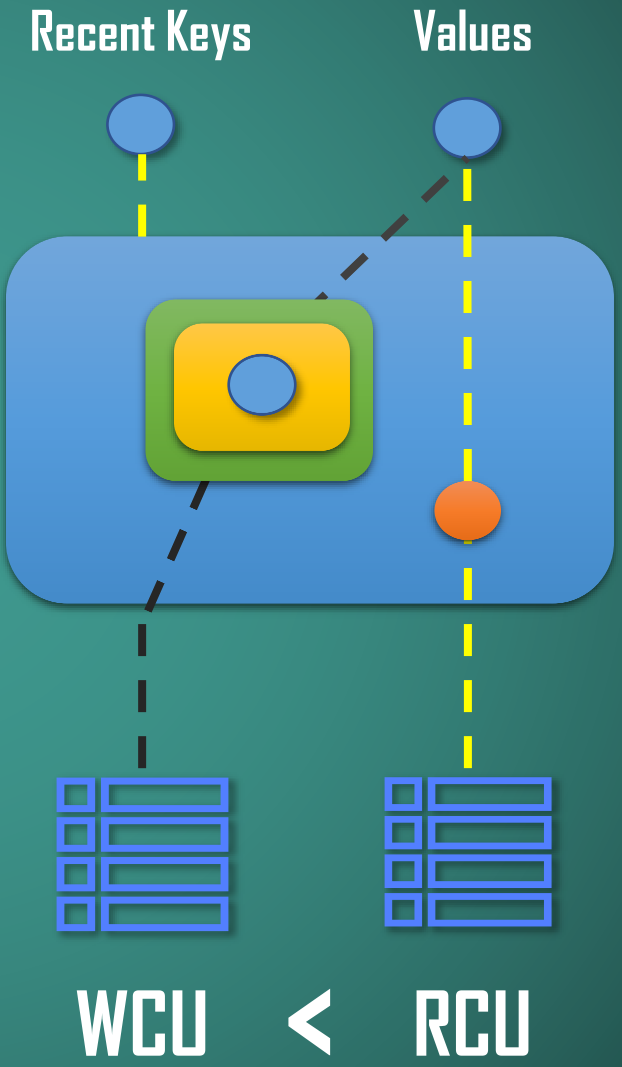
```
public LinkedBlockingQueue() {
    this(Integer.MAX_VALUE);
}
```



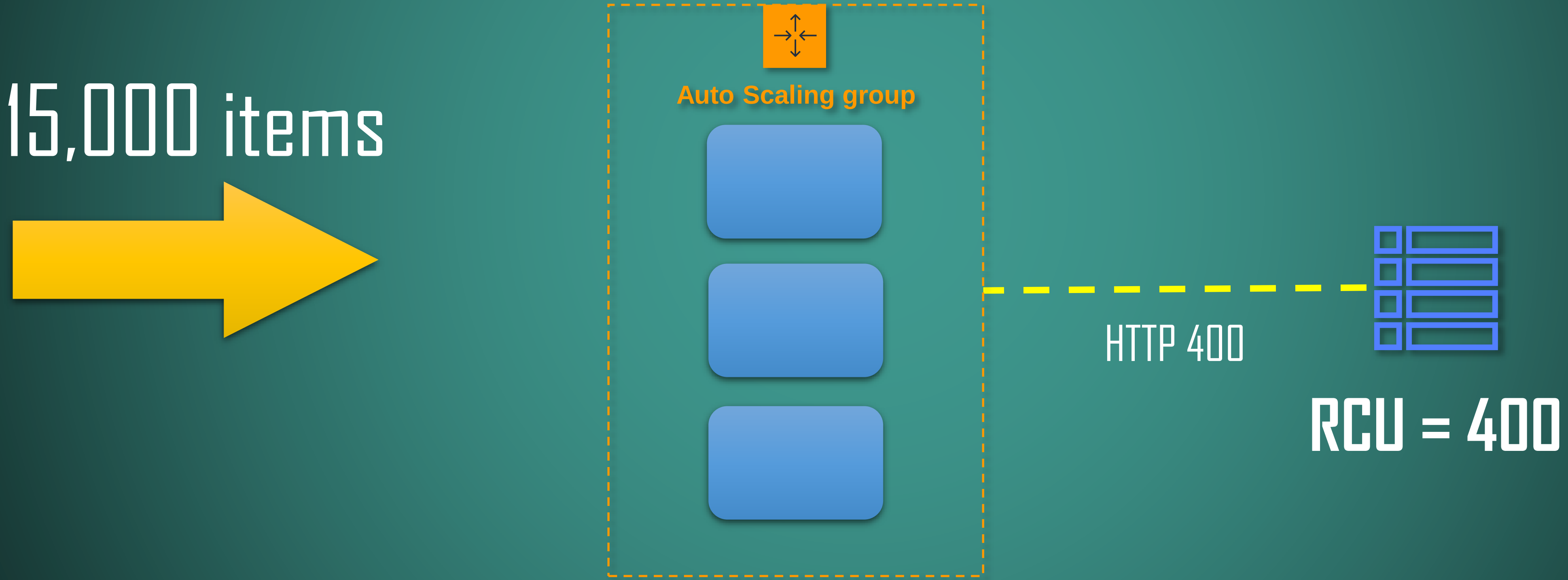
Separate Critical Path

Circuit Breaker

+ Bulkhead



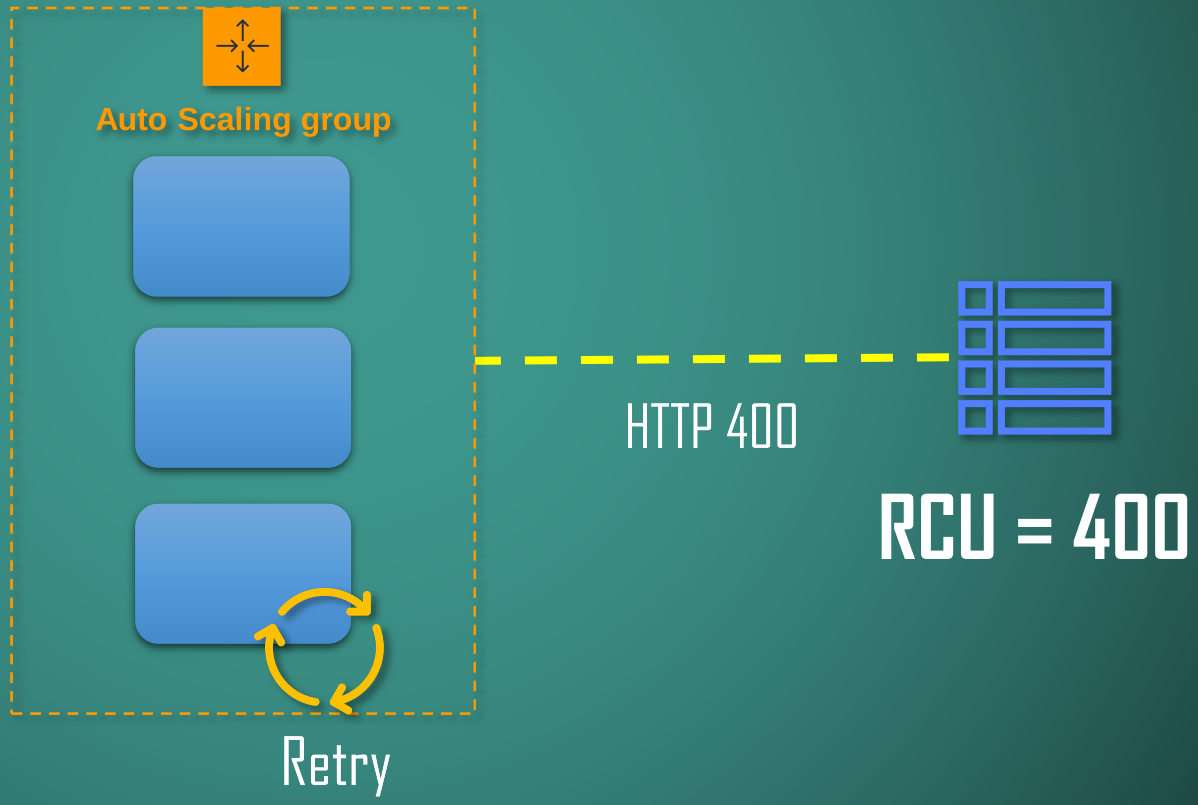
Baseline



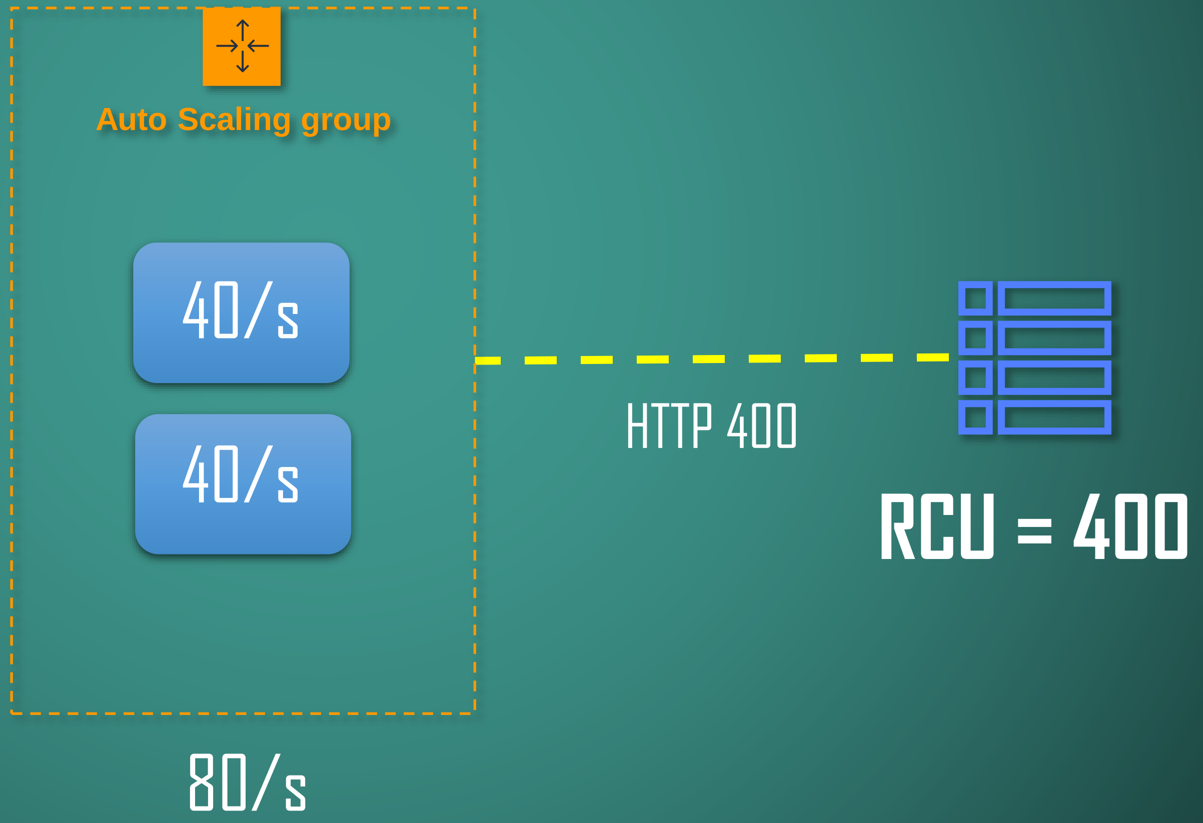
<epam>

Retries

🚫 No Fairness

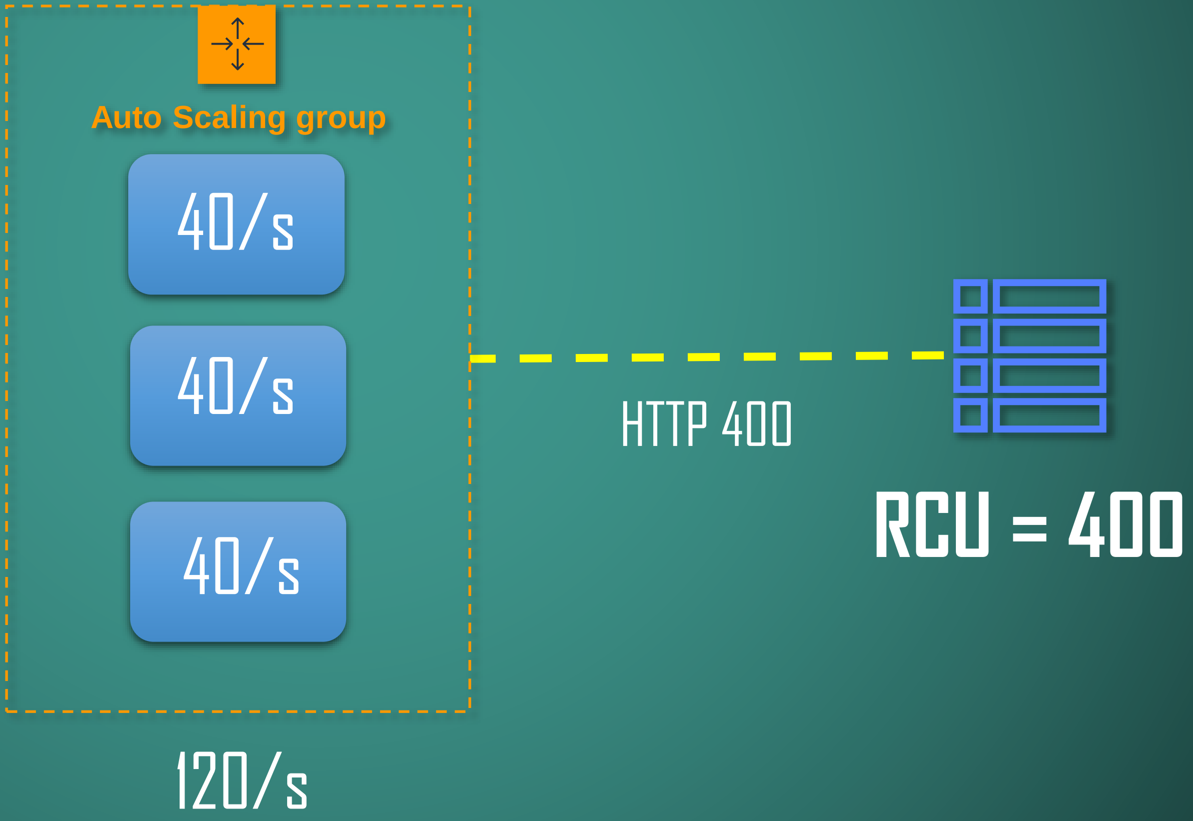


Built In Rate-Limiting



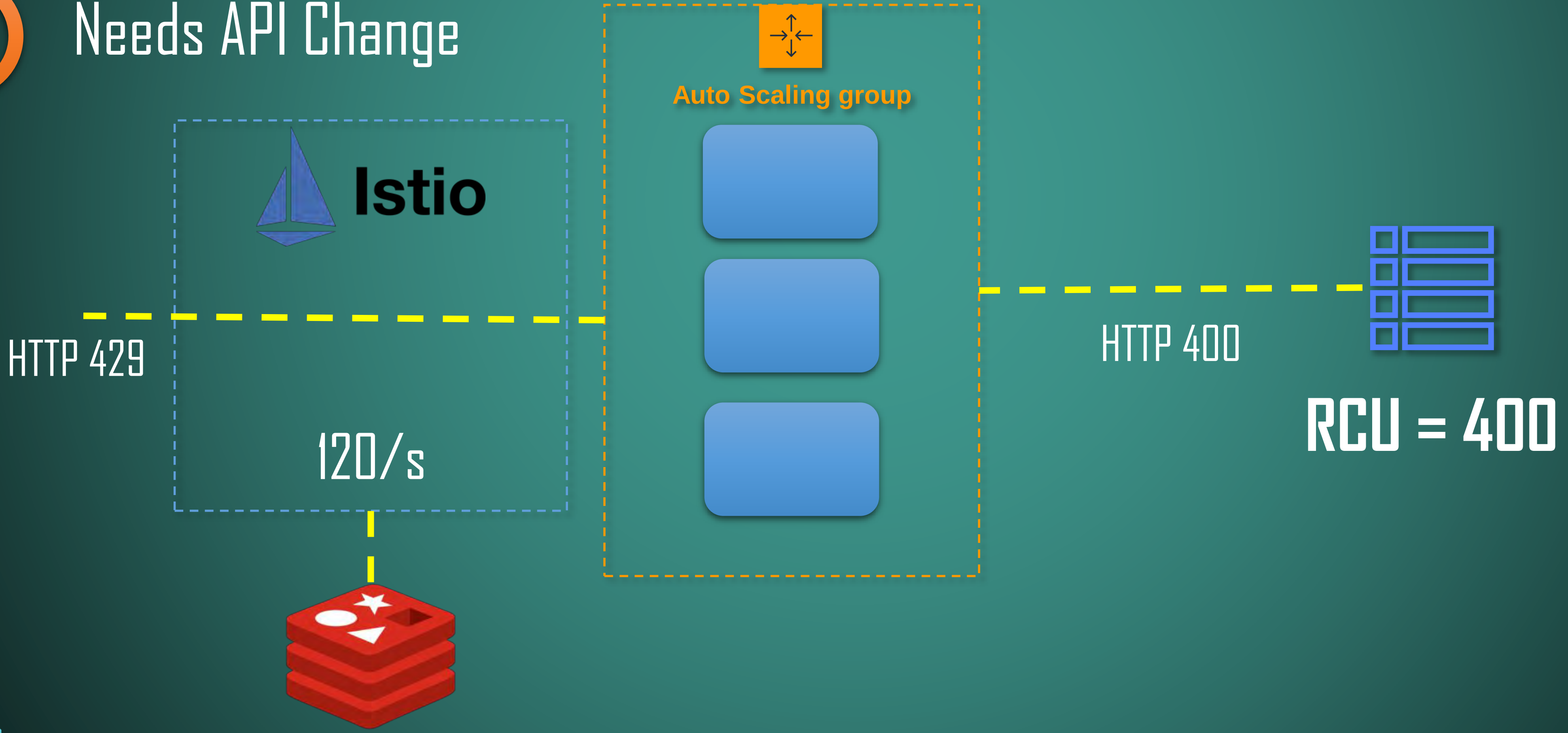
Built In Rate-Limiting

🚫 Rate-Limit Changes

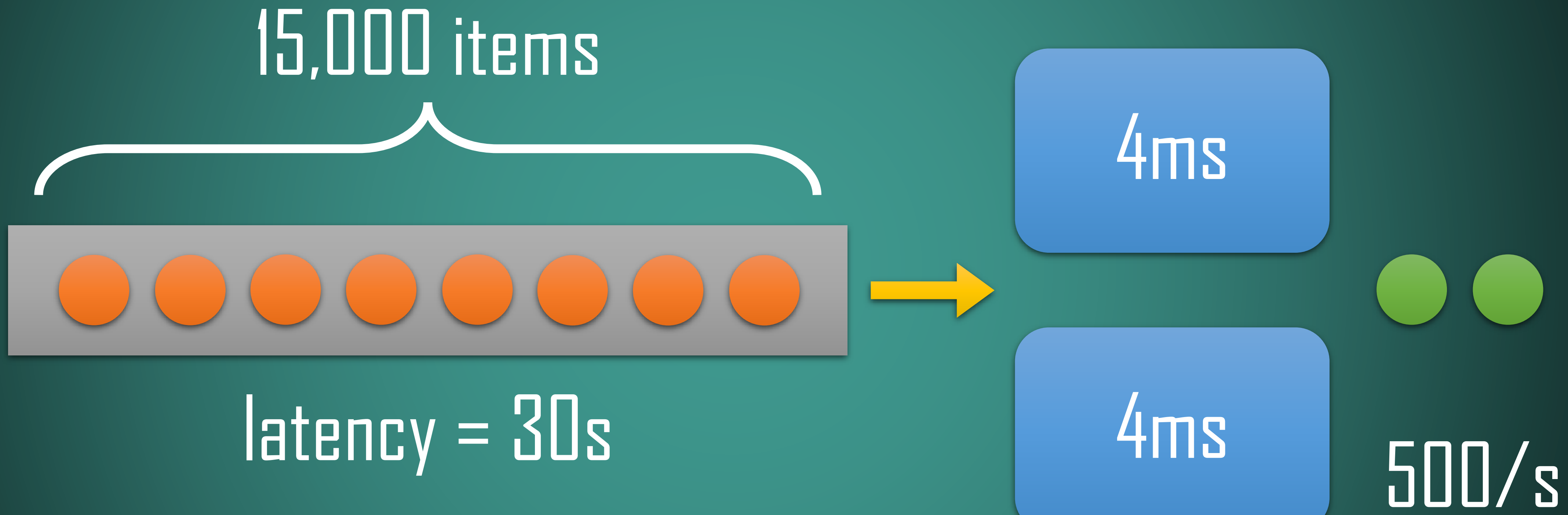


Service Mesh

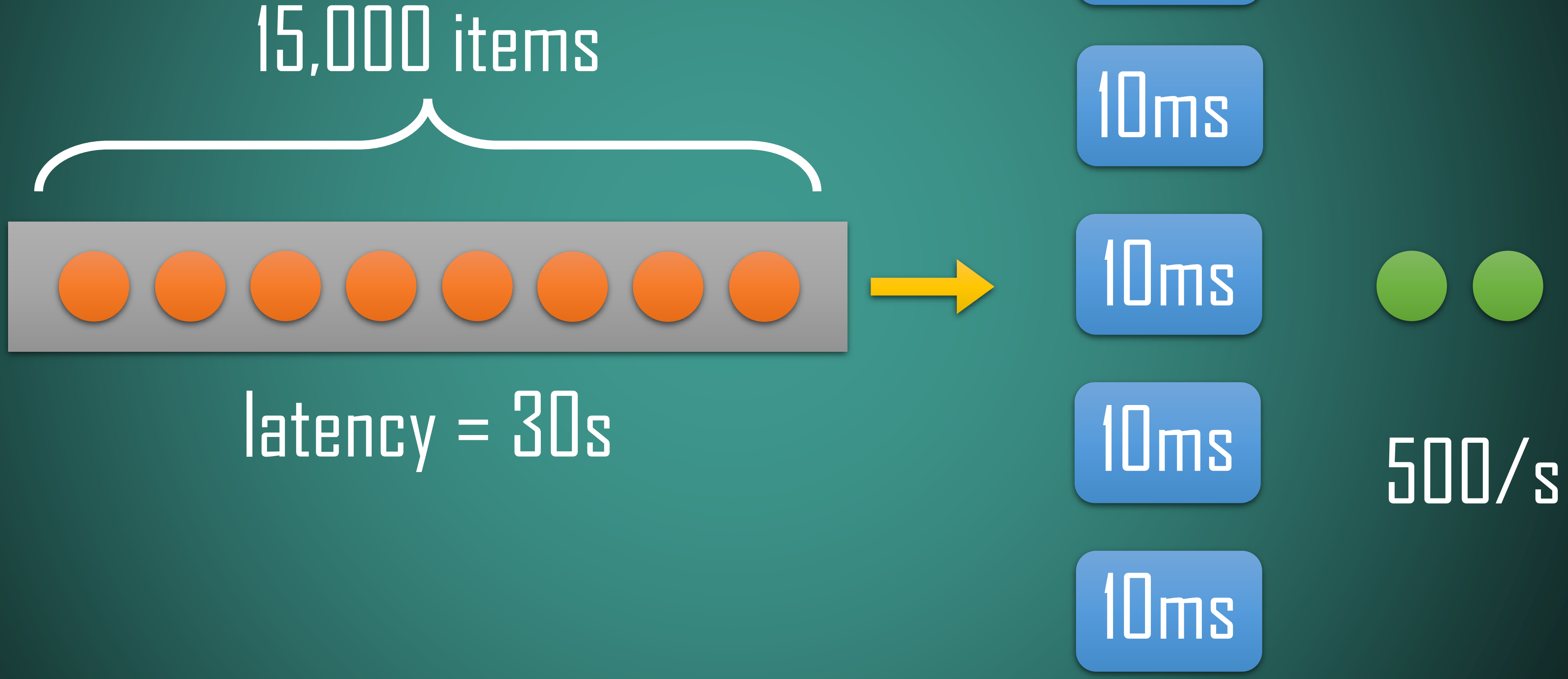
Needs API Change



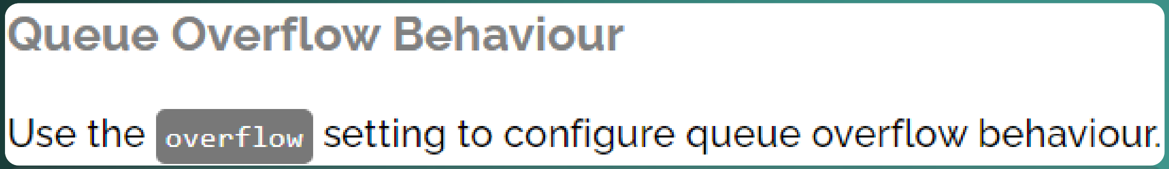
Queuing Theory



Queuing Theory

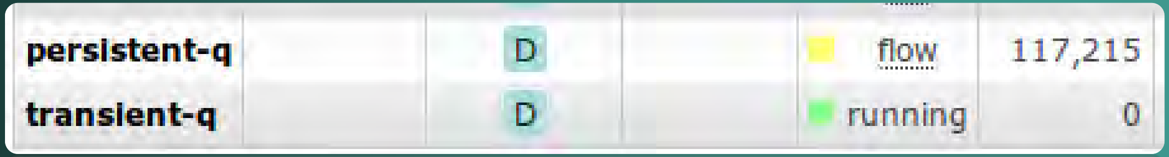


Moving Towards RabbitMQ

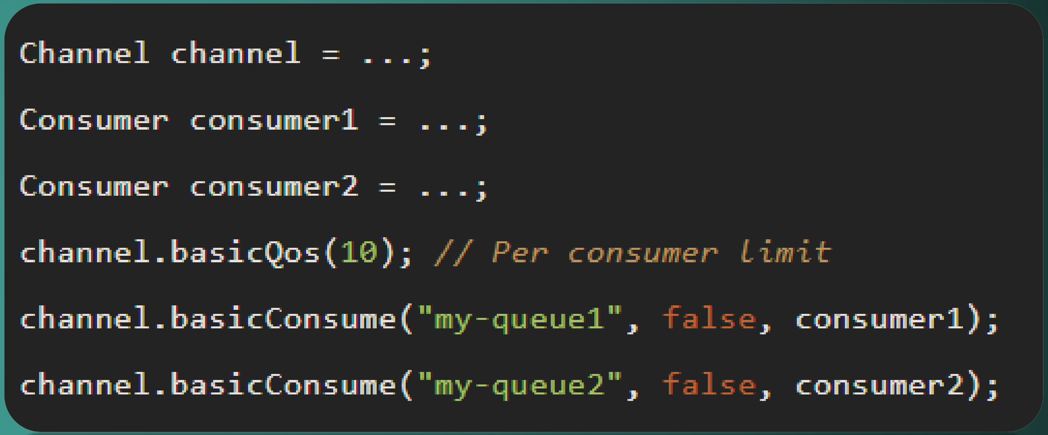


Queue Overflow Behaviour

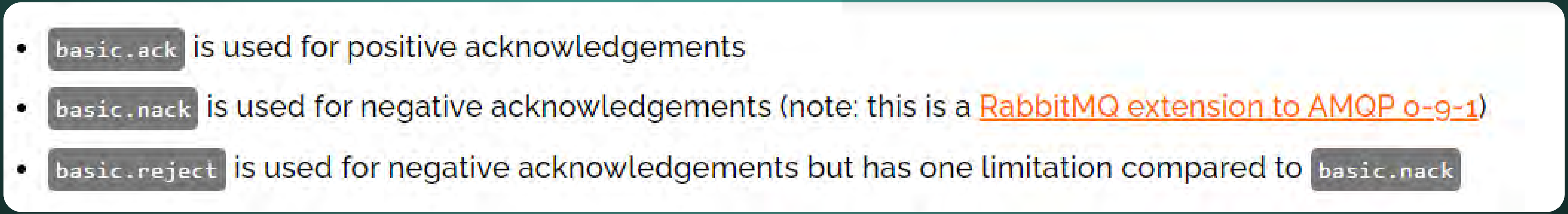
Use the `overflow` setting to configure queue overflow behaviour.



persistent-q		D		🟨 flow	117,215
transient-q		D		🟩 running	0

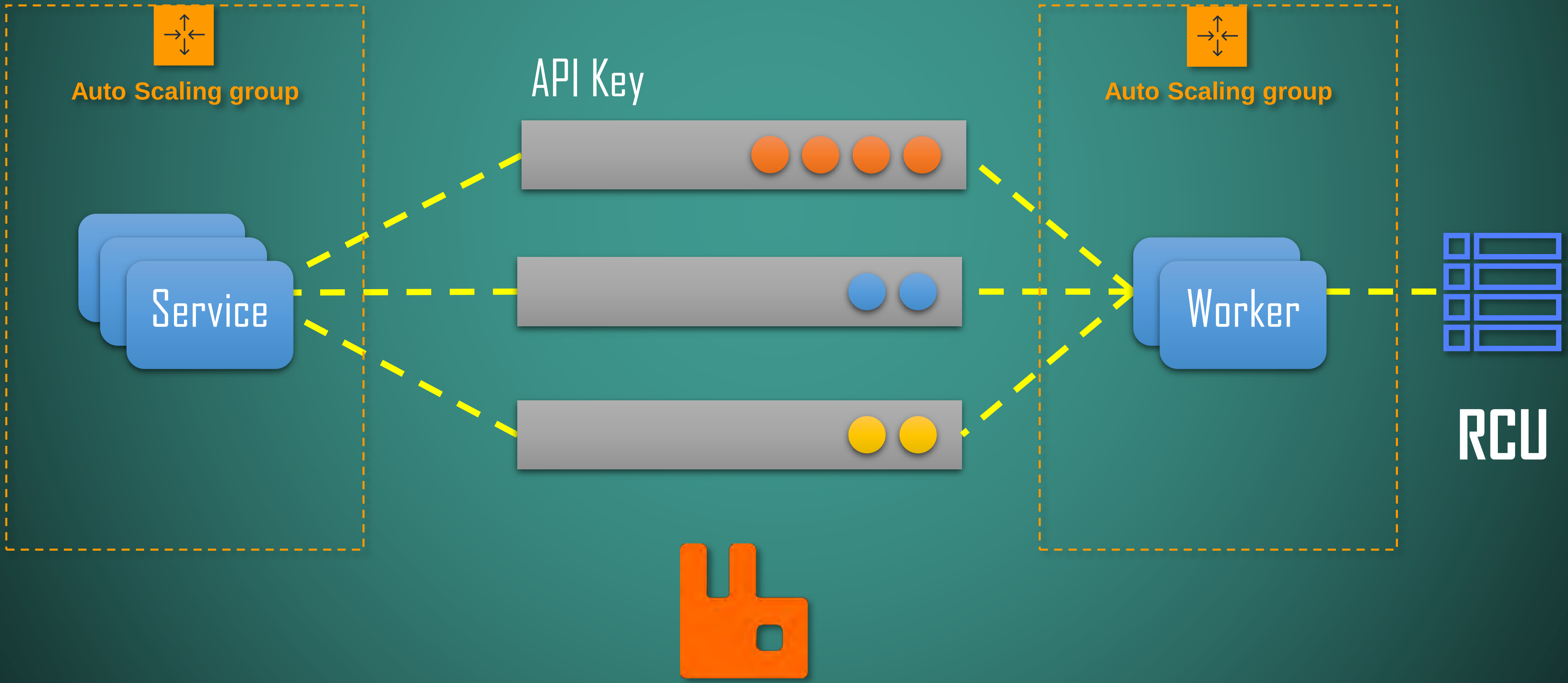


```
Channel channel = ...;
Consumer consumer1 = ...;
Consumer consumer2 = ...;
channel.basicQos(10); // Per consumer limit
channel.basicConsume("my-queue1", false, consumer1);
channel.basicConsume("my-queue2", false, consumer2);
```

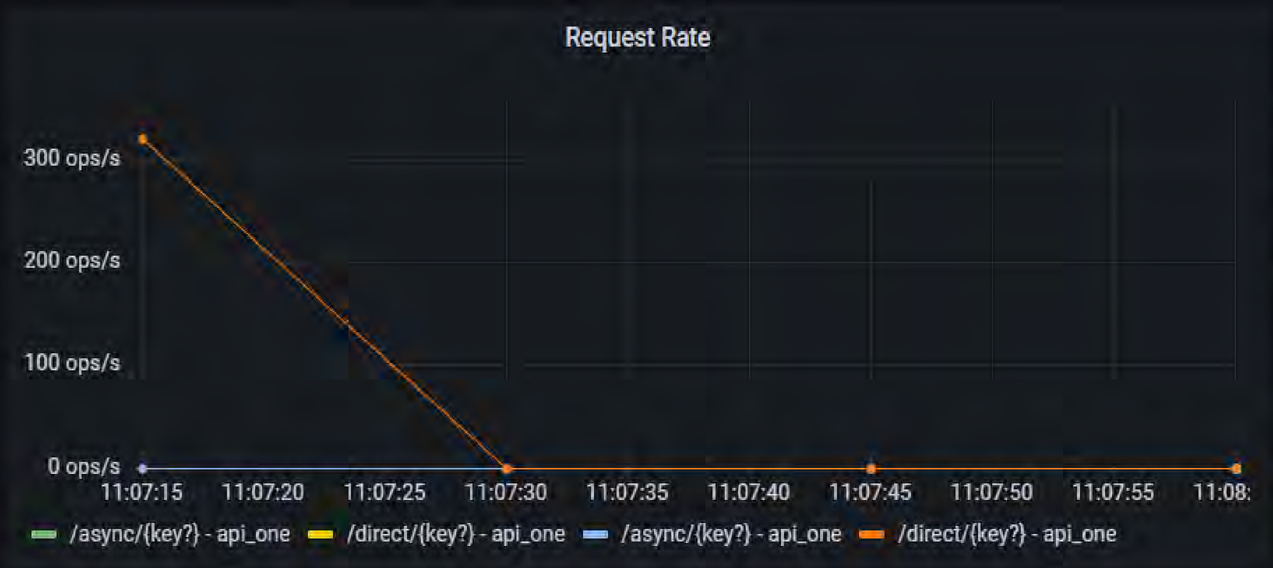


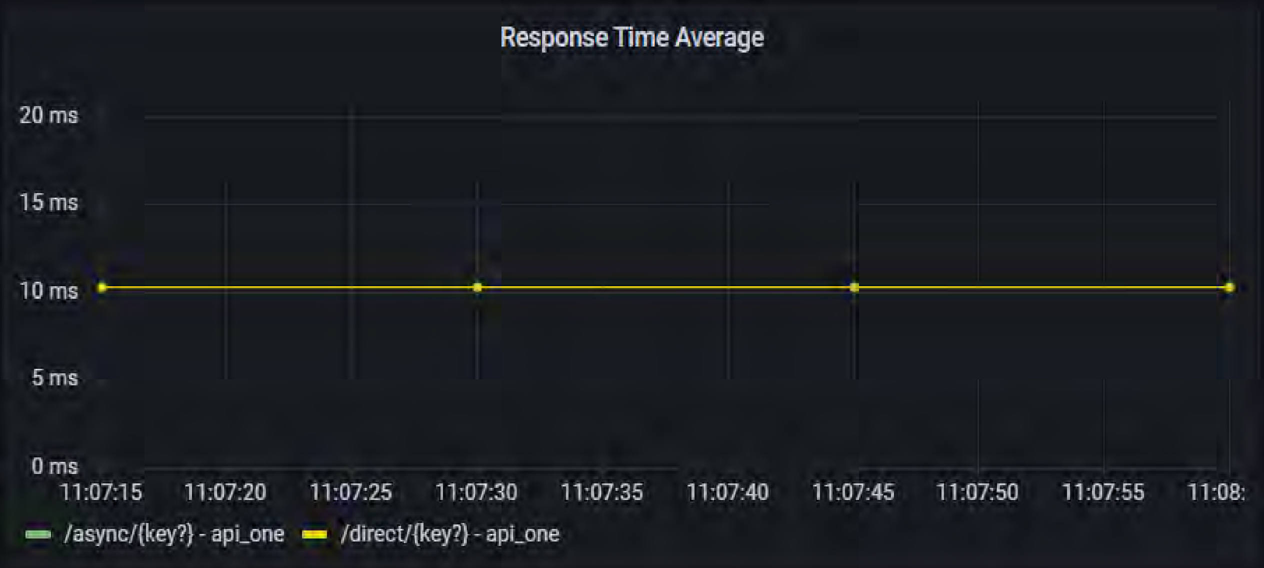
- `basic.ack` is used for positive acknowledgements
- `basic.nack` is used for negative acknowledgements (note: this is a RabbitMQ extension to AMQP 0-9-1)
- `basic.reject` is used for negative acknowledgements but has one limitation compared to `basic.nack`

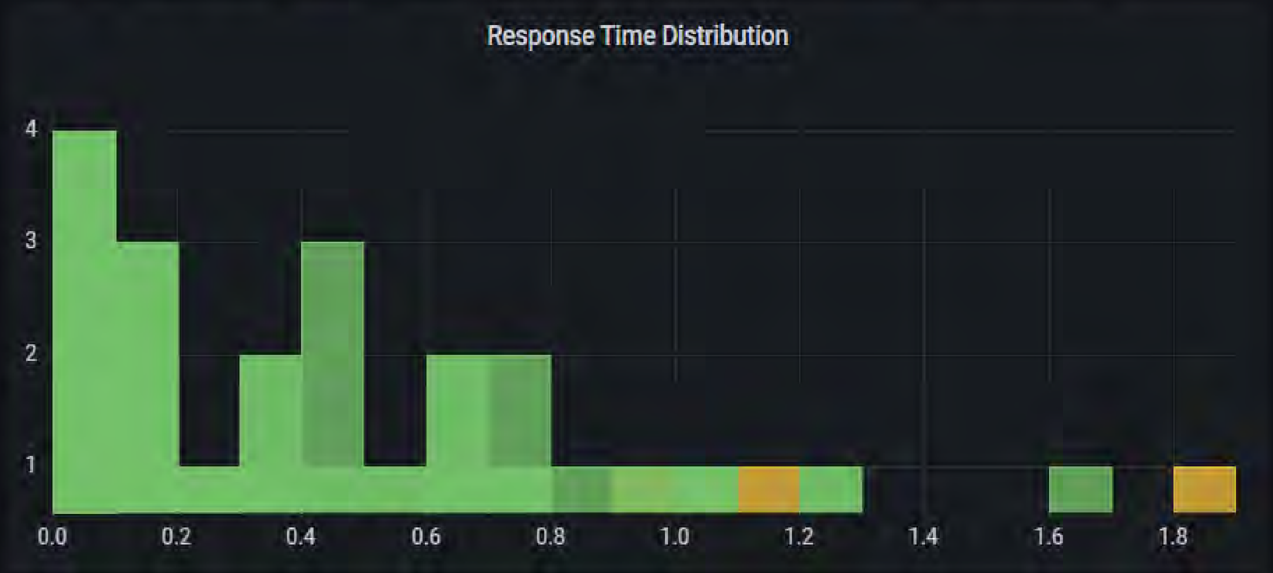
Separate by API Keys



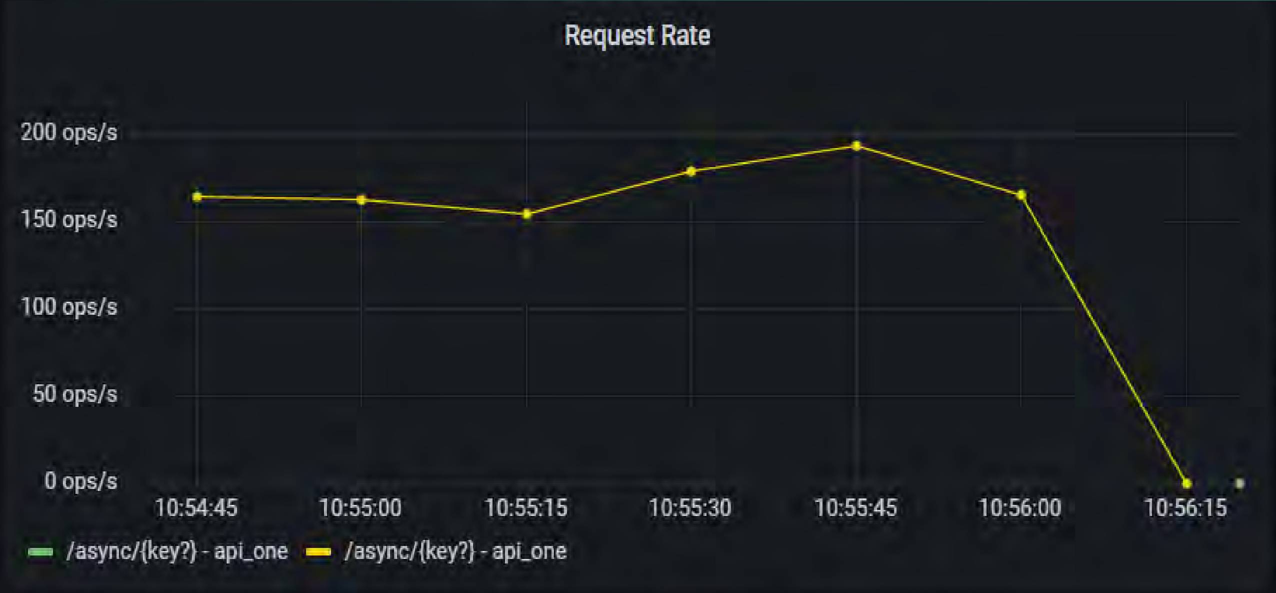
Direct Read

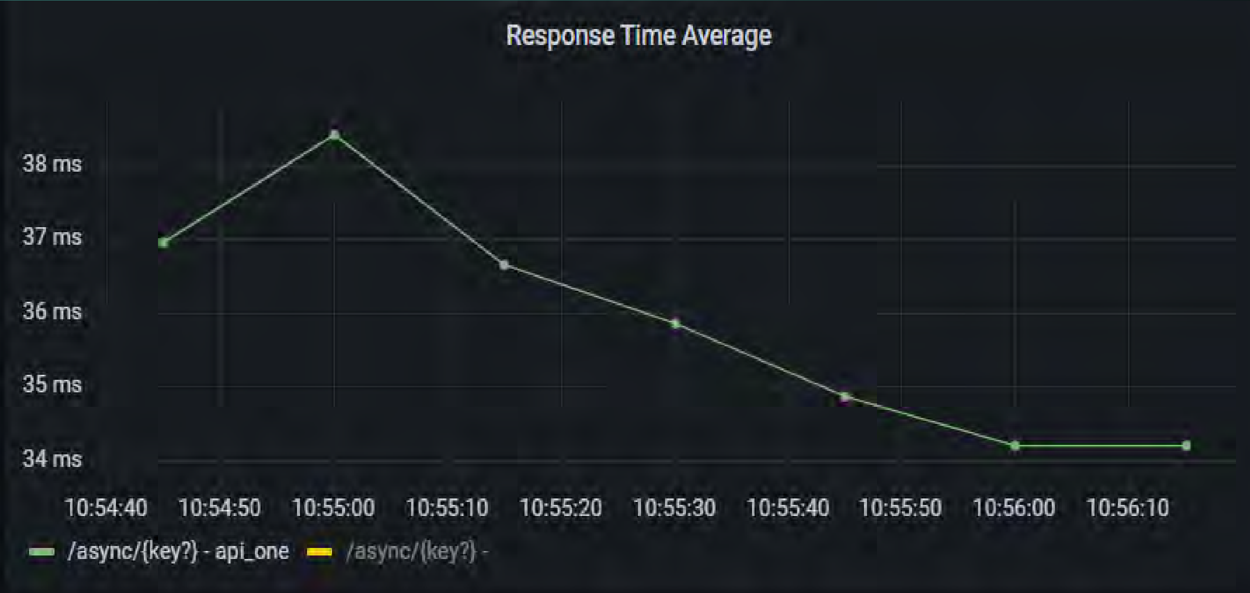

Request Rate
300 ops/s
200 ops/s
100 ops/s
0 ops/s
11:07:15
11:07:20
11:07:25
11:07:30
11:07:35
11:07:40
11:07:45
11:07:50
11:07:55
11:08:
/async/{key?} - api_one
/direct/{key?} - api_one
/async/{key?} - api_one
/direct/{key?} - api_one

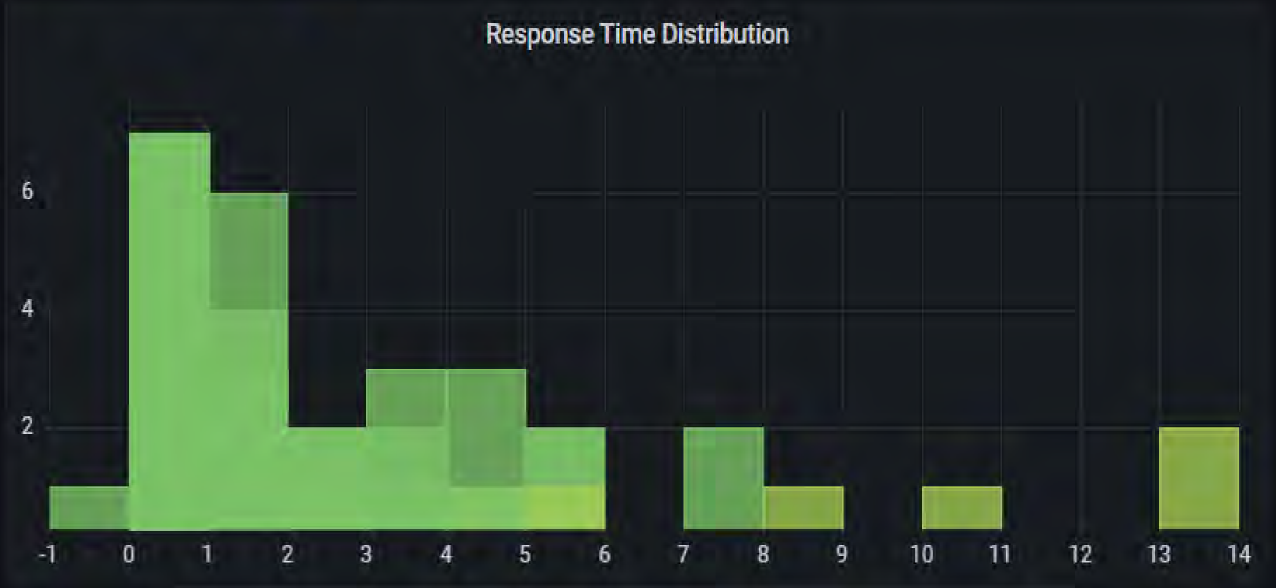

Response Time Average
20 ms
15 ms
10 ms
5 ms
0 ms
11:07:15
11:07:20
11:07:25
11:07:30
11:07:35
11:07:40
11:07:45
11:07:50
11:07:55
11:08:
/async/{key?} - api_one
/direct/{key?} - api_one


Response Time Distribution
4
3
2
1
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1.8

Single Worker

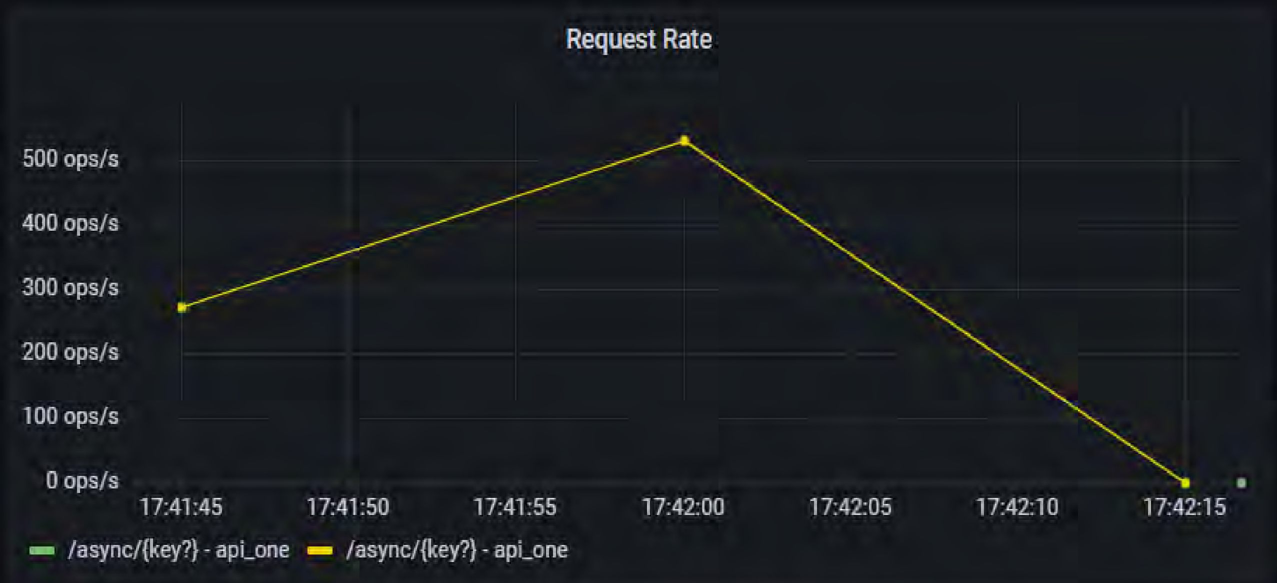


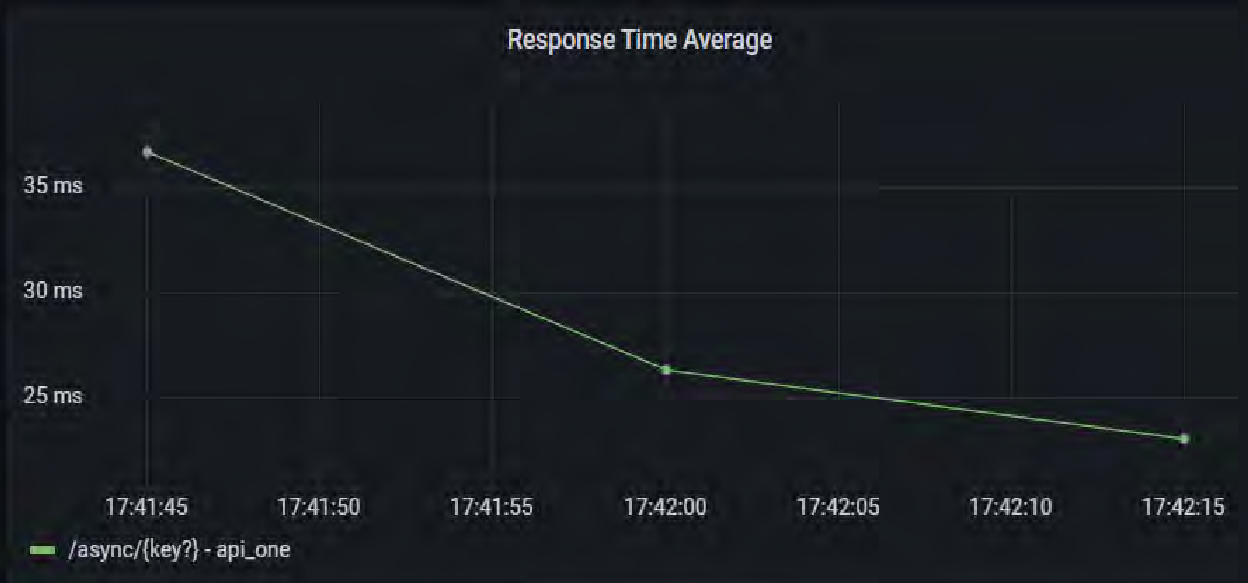


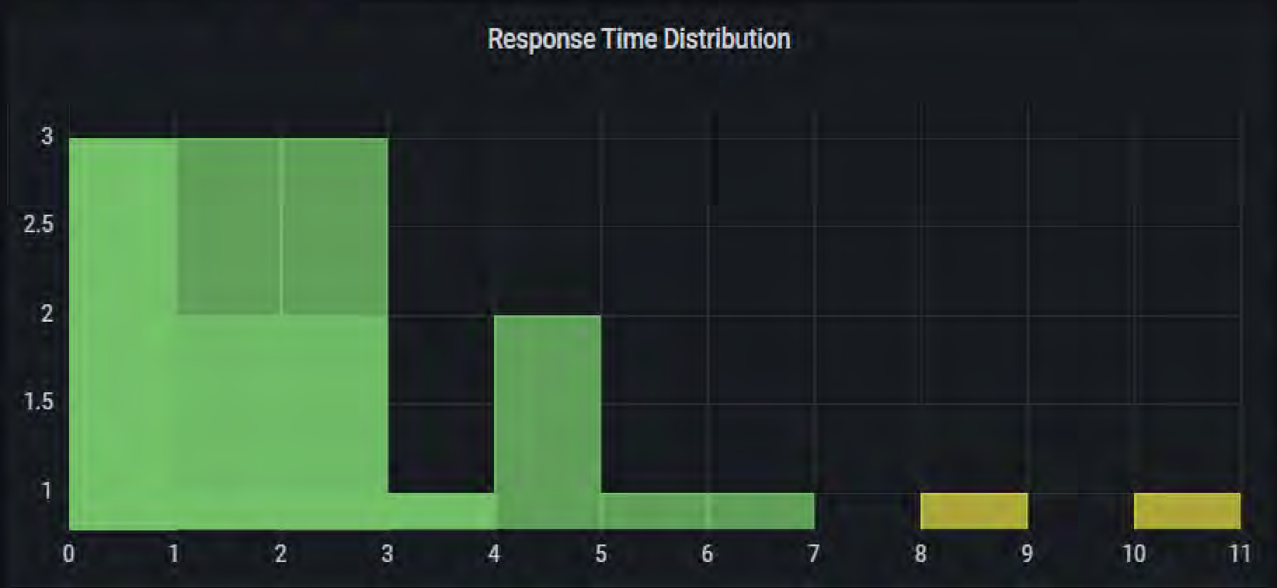


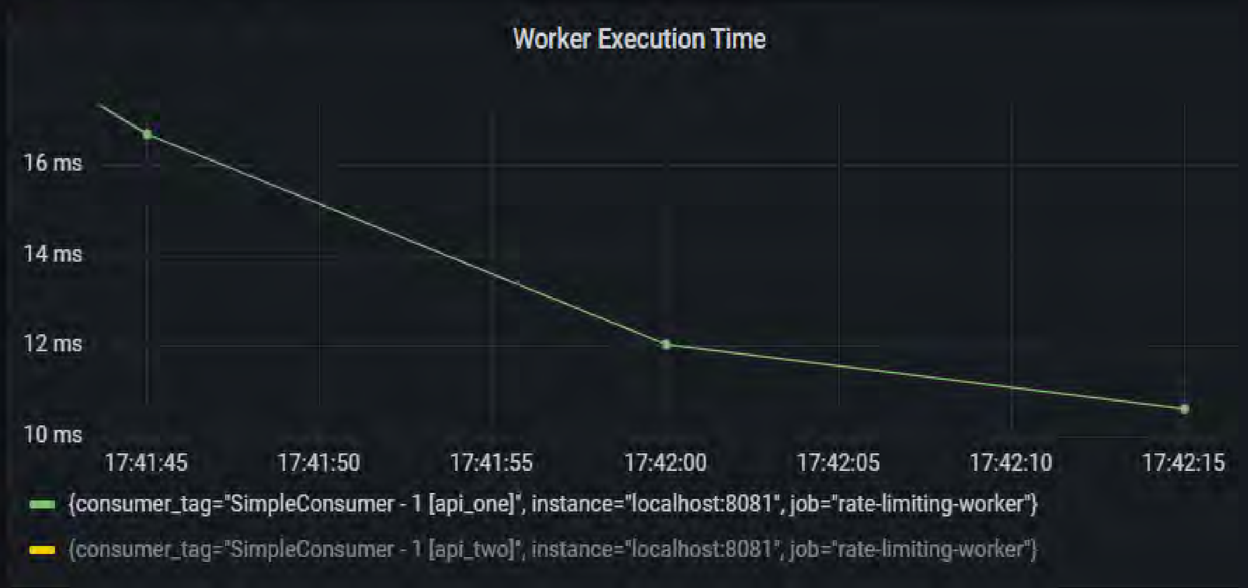
<epam>

Five Workers









Thank You!

ORESZTÉSZ MARGARITISZ

ASSOCIATE CHIEF SOFTWARE ENGINEER

@gitaroktato

gitaroktato

https://www.linkedin.com/in/oresztesz

References

Latency
https://www.igvita.com/2012/07/19/latency-the-new-web-performance-bottleneck/
https://docs.google.com/a/chromium.org/viewer?a=v&pid=sites&srcid=Y2hyb21pdW0ub3JnfGRldnxneDoxMzcyOWI1N2I4YzI3NzE2
https://www.cloudping.co/
https://aws-latency-test.com/
https://colin-scott.github.io/personal_website/research/interactive_latency.html
https://gist.github.com/jboner/2841832
https://blog.bytebytego.com/p/ep22-latency-numbers-you-should-know

Availability
https://github.com/gitaroktato/microservices-availability-simulator
https://eventhelix.com/fault-handling/system-reliability-availability
https://eventhelix.com/fault-handling/reliability-availability-basics

https://en.wikipedia.org/wiki/Fallacies_of_distributed_computing

References

Backpressure with SQS
https://d1.awsstatic.com/architecture-diagrams/ArchitectureDiagrams/unreal-engines-pixel-streaming-on-aws-ra.pdf?did=wp_card&trk=wp_card

Reliability
https://www.joelonsoftware.com/2002/11/11/the-law-of-leaky-abstractions/

Queuing Theory
https://www.youtube.com/watch?v=oQGreeij-0E
https://www.youtube.com/watch?v=raRpbsWQBCo
https://en.wikipedia.org/wiki/Little%27s_law
https://en.wikipedia.org/wiki/Kendall%27s_notation
https://dzone.com/articles/applying-back-pressure-when

Rapid-Read Protection
http://www.datastax.com/dev/blog/rapid-read-protection-in-cassandra-2-0-2

References

Architecture Katas
https://nealford.com/katas/kata?id=GoingGoingGone

Kafka / Kinesis Multi-Region Examples
https://aws.amazon.com/blogs/big-data/increase-apache-kafkas-resiliency-with-a-multi-region-deployment-and-mirrormaker-2
https://aws.amazon.com/blogs/big-data/build-highly-available-streams-with-amazon-kinesis-data-streams

The Twelve-Factor App
https://12factor.net/

Rate-Limiting Sandbox
https://github.com/gitaroktato/system-design-excercises/tree/main/rate-limiting

Microservices Availability Simulator
https://github.com/gitaroktato/microservices-availability-simulator

References

Backpressure with RabbitMQ
https://www.rabbitmq.com/maxlength.html
https://www.rabbitmq.com/tutorials/tutorial-six-java.html
https://www.rabbitmq.com/consumers.html#single-active-consumer
https://www.rabbitmq.com/flow-control.html
https://blog.rabbitmq.com/posts/2020/05/quorum-queues-and-flow-control-the-concepts
https://www.rabbitmq.com/consumer-prefetch.html
https://blog.rabbitmq.com/posts/2014/04/finding-bottlenecks-with-rabbitmq-3-3/
https://blog.rabbitmq.com/posts/2015/10/new-credit-flow-settings-on-rabbitmq-3-5-5/
https://www.rabbitmq.com/confirms.html#publisher-confirms